International Journal of Engineering Trends and Technology
ISSN: 2231–5381 / https://doi.org/10.14445/22315381/IJETT-V72I7P137

Volume 72 Issue 7, 339-349, July 2024
© 2024 Seventh Sense Research Group®

Original Article

Energy Optimization in Software Development: A Comparative Study of Sorting Techniques

P.S. Felix[1], M. Mohankumar[2]

[1,2]*Department of CS, Karpagam Academy of Higher Education, Coimbatore, Tamil Nadu, India.*

[1]*Corresponding Author : psfelix@gmail.com*

Received: 16 March 2024 Revised: 10 June 2024 Accepted: 01 July 2024 Published: 26 July 2024

Abstract *- The escalating levels of greenhouse gas emissions are attributable to human activities that fueled a pressing need to address energy consumption across various sectors, particularly in the realm of software development. This article explores the imperative of energy efficiency within the context of Green Software development, emphasizing its significance in mitigating environmental impact. Focusing on the comparative analysis of two widely-used sorting algorithms, Bubble Sort and Quick Sort, those study investigates their energy efficiency when handling large numerical datasets. The methodology encompasses meticulous steps, including application selection, data generation, power measurement, energy consumption analysis, and report generation. Through rigorous experimentation and analysis, the research article elucidates the energy consumption patterns of the sorting algorithms, providing insights into optimizing energy usage in software development. The findings underscore the importance of developing energy-efficient software systems, aligning with principles of environmental sustainability and responsible technological innovation.*

Keywords *- Green software development, Energy efficiency, Energy consumption analysis, Green metrics, Sustainable software engineering.*

1. Introduction

Greenhouse gases, predominantly amplified by human activities in the past 150 years, capture heat and play a significant role in global warming. The primary source of these emissions in the United States is the combustion of fossil fuels for electricity, heating, and transportation, as shown in Fig 1. In 2020, electricity generation, responsible for 25% of greenhouse gas emissions and heavily dependent on fossil fuel combustion, was the second-highest contributor. Conversely, India has established lofty climate objectives. By 2030, it intends to decrease its anticipated carbon emissions by 1 billion tonnes and augment its non-fossil energy volume to 500 gigawatts. In addition, India aspires to fulfil 50% of its energy requirements from renewable sources by 2030 and attain Net Zero emissions by 2070. Given the urgent need to curb significant electricity usage, one potential solution is to decrease energy consumption in data centres where software applications are running on a huge scale. The development of Green Software places a strong emphasis on energy efficiency, leading to a comparative study of two popular sorting algorithms, Quick Sort and Bubble Sort, with a focus on their energy efficiency when dealing with large numerical datasets. In previous studies, the utilization of CPU, memory, and storage disk was recorded. Nevertheless, to discern the impact of choosing the right sorting algorithms, it's essential to determine the exact energy consumption values.This research covers several aspects, including the creation of a controlled testing environment, data preparation, the use of power measurement tools, test execution, and the subsequent analysis of energy consumption patterns. The discussion also explores code optimization techniques and the importance of green metrics, underscoring the necessity to develop software systems that not only provide functionality and performance but also prioritize energy efficiency and environmental sustainability. The subsequent sections of this article are arranged the following way; Section II presents a literature review. The methodology is discussed in Section III, related studies in Section IV, and the approach in Section V.Section VI details the experimental design and validation, which is based on the approach and its execution. The outcomes of our experiment are discussed in detail in Section VII. Finally, Section VIII provides a conclusion and potential directions for future research.

2. Literature Review

2.1. Software Development

Green Software, often described as a development methodology that caters to present requirements while ensuring the ability of future generations to fulfil their needs, has garnered considerable attention. This notion is an integral component of the more extensive Green IT initiative, which emphasizes the sustainability facets of Data Centres. A

This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/)

plethora of publications underscores these elements, underscoring the topic's significance and pertinence. As shown in Figure 2, from 1990 to 2010, developers primarily focused on software requirements and development processes. [12] They leveraged technology for various purposes, including design safety and reliability, testing at different development levels, software safety analysis, hazard analysis, and obtaining certifications and standard resources. These measures were aimed at improving the quality and reliability of software products. Green metrics in software development is another important topic [5]. Green metrics are measures that help quantify the environmental impact or energy efficiency of a software system. They provide tangible data that can guide developers in creating more sustainable software. The field also includes the development and use of energy-aware tools and techniques [6]. These are specialized energy consumption of software. They can be integral in creating software that is energy efficient.

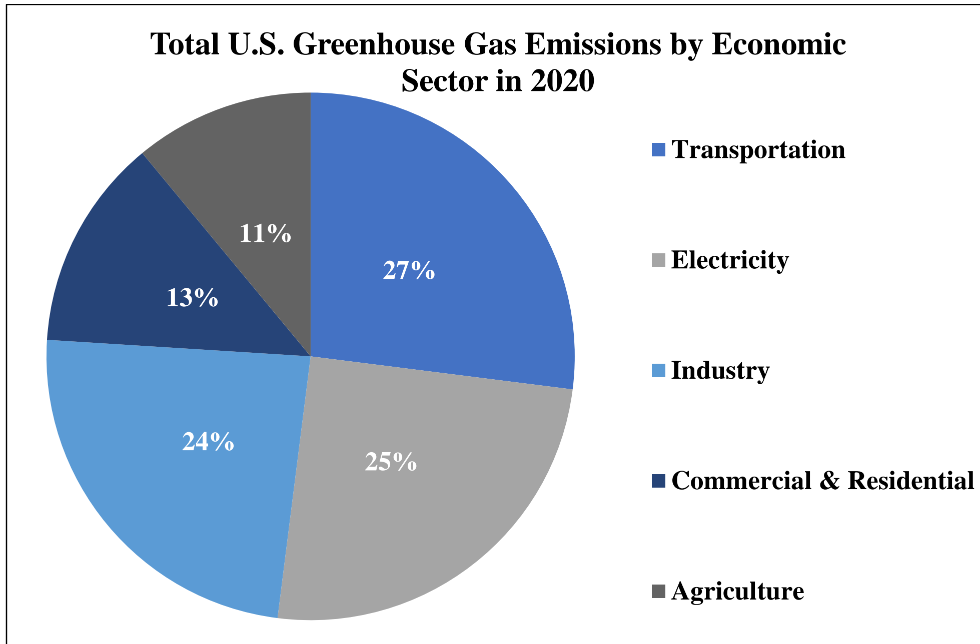

Fig. 1 Total emissions in 2020 in the USA

Source: U.S. Environmental Protection Agency (2020)

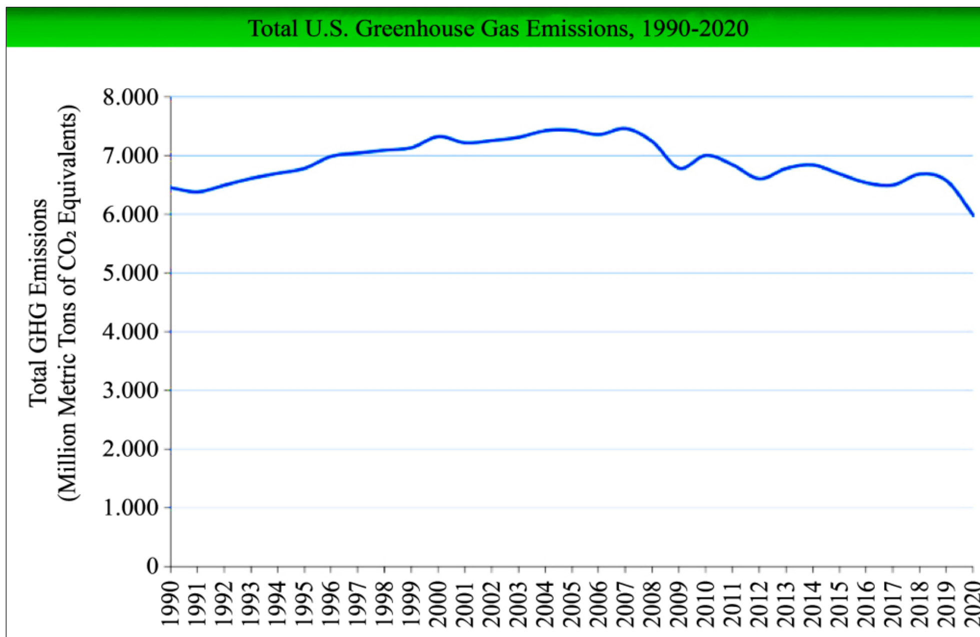

Fig. 2 All emission estimates from 1990 to 2020

Source: U.S. Environmental Protection Agency (2022)

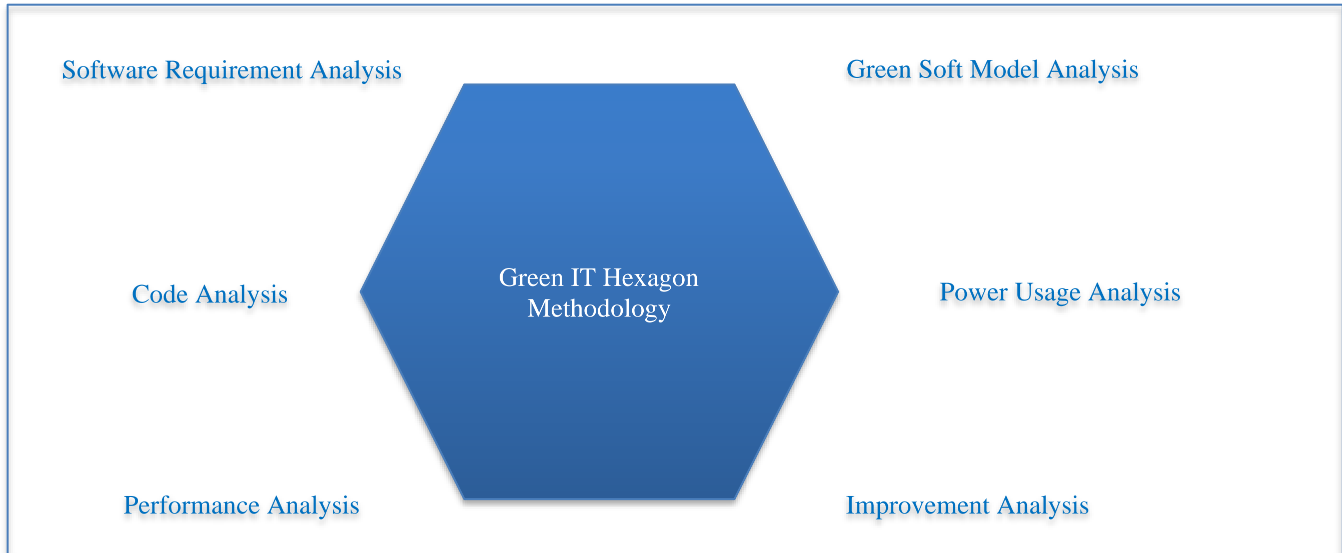

Fig. 3 Green IT hexagon methodology diagram

2.2. Software Testing

Software testing, a critical component of ensuring quality, has revealed a substantial efficiency gap. This gap is observed between the codes produced by experienced developers and those generated by automatic interpreters [13], suggesting that human expertise and intuition in coding still play a vital role. However, despite these stringent measures, Green IT approaches are not universally adopted. Approximately one-third of European organizations do not follow these approaches, [11] while only about 20% of firms regularly implement energy consumption regulatory measures. This indicates a need for greater awareness and implementation of Green IT practices.

2.3. Software Sustainability

In response to this, researchers like Mahaux et al. [2] and Becker et al. [3] have taken a step back to redefine the objective of software sustainability. They emphasized the importance of software engineering and the design of sustainable application software, suggesting a shift in focus from merely creating functional software to developing software that is both functional and sustainable.

2.4. Energy Optimization

Energy optimization in the realm of software development is a comprehensive field encompassing a variety of subtopics. One such subtopic is the examination of hardware and software strategies for energy-conserving computation. This includes the study of both hardware and software components that can contribute to more energy-saving computing. Another related area is sustainable software engineering. [4] This discipline incorporates sustainability principles into all facets of software engineering practices. It aims to create software solutions that are not only efficient and effective but environmentally friendly. Modelling and optimization of energy systems is another crucial area [7]. It involves creating detailed models of energy systems and using various optimization techniques to improve their efficiency. This can lead to significant energy savings in software systems.

2.5. High Performance Computing

The European Union's Horizon 2020 research program funds the Software Development Toolkit for Energy Optimization and Technical Debt Elimination project [10]. This project is designed to reduce the cost, development time, and complexity associated with low-energy software development processes. Energy consumption in High Performance Computing (HPC) infrastructures is a key topic [8]. HPC infrastructures are known for their high energy consumption and finding ways to reduce power consumption, which can lead to significant energy savings.Lastly, energy consumption in cloud-based data centres is a major concern [9]. Data centres that provide cloud-based services consume a significant amount of energy. Finding ways to make these data centres greener is an important area of research in energy optimization in software development.

3. Methodology

The Green IT Hexagon methodology shown in Fig 3 serves as a comprehensive approach for the assessment of green software. This methodology is designed with the primary objective of enhancing Software Energy Usage, thereby promoting more sustainable practices in the field of Information Technology.The methodology operates on the principle that the metrics for energy-efficient software are contingent on the useful work performed by the software. Given the complexity of modern software, which comprises numerous modules, each serving a unique purpose, the methodology acknowledges that there may be more than one

applicable metric. In the Green IT Hexagon methodology, these software parts can be evaluated either individually or in combination. This flexibility allows for a more nuanced understanding of the software's energy usage patterns and efficiency. However, for an accurate comparison of different software, it is recommended that the measured modules be as similar as possible. At its core, the methodology proposes a generic metric for software energy efficiency. This metric, although not explicitly defined here, serves as a standard measure that can be applied across different software systems to assess their energy usage and efficiency.

This approach facilitates a more standardized and objective evaluation of software energy usage [14], thereby supporting the broader goal of promoting energy efficiency and sustainability in software development. The commonly used approach to measure energy efficiency is as below.

$$Energy\ Efficiency = \frac{Useful\ Work\ Done}{Used\ Energy}$$

4. Related Work

4.1. Sorting Techniques and Algorithms

Sorting techniques and algorithms are core principles in computer science, designed to order data in a specific sequence. These strategies range in their complexity and efficiency, with some being more applicable to smaller data sets and others optimized for larger ones [1]. The selection of a suitable sorting technique can have a significant impact on the performance of a software program, making it essential to understand these strategies' characteristics. They play a crucial role in various computing fields, including data analysis, machine learning, and software engineering. Mastery of these strategies is vital for effective data management and problem-solving.

4.2. Energy Monitoring

Energy monitoring policies for observing a program's energy usage can be categorized into two types: external and internal evaluators. External energy monitors employ tools like voltmeters and ammeters to evaluate the system as a whole. However, their ability to monitor individual programs is limited as they lack the granularity to identify energy usage at the component level. On the other hand, internal evaluators are integrated within the system, like the Power Reading Unit (PRU) for power monitoring. They measure energy registers, process wakeups, and CPU state transitions to provide a more detailed view of a program's energy consumption.

4.3. Optimization of Code

In compiler design, code optimization is a technique of program transformation that strives to enhance the intermediate code by reducing its resource consumption, thereby leading to machine code that runs more swiftly. Dead code elimination is a technique used in optimizing code where sections of code that are never executed during runtime are identified and removed. This can enhance program efficiency, improve maintainability, and reduce program size.

Here's a simple example of dead code elimination and code optimization in C#:

```
public int Calc(int val1, int val2)
{
    int z = val1 * val2; // Dead code, z is
never used
    int m = 10;
    int n = 20;
    int result = m + n; // To be Optimized
    return result;
}
```

In the above code, `z` is never used in the function, so it can be considered dead code and can be eliminated. The variables `m` and `n` are constants, so the expression `m + n` can be evaluated at compile time, which is a form of code optimization known as constant folding. The optimized code is shown below.

```
public int Calc(int val1, int val2)
{
    int result = 30; // Optimized code
    return result;
}
```

In this optimized code, the dead code has been eliminated, and the expression has been evaluated at compile time. This results in a more efficient program execution.

4.4. Energy Model

The energy consumption of a program [14] in an application model can be broken down as follows:

$$E_{\text{application}} = E_{\text{active}} + E_{\text{wait}} + E_{\text{idle}}$$

Here, E_{active} represents the running time of the application, E_{wait} is the waiting time for other components, and E_{idle} is the time during which the system is not performing any work for the specific application.

5. Approach

The approach outlined in Figure 4 for analysing the energy efficiency of two sorting algorithms follows a systematic and structured process aimed at optimizing energy usage during software development. Here's a detailed elaboration of each step:

5.1. Application Selection

The first step involves selecting the specific application; in this case, both the sorting algorithms Quick Sort and Bubble Sort were considered for the testing process. Despite producing the same result, these algorithms perform differently when processing large volumes of data. This selection is crucial as it defines the experiment's scope and focus. The algorithms for evaluation are chosen based on their relevance, popularity, and potential influence on energy consumption. Through the selection of these algorithms, a targeted strategy to optimize energy usage in software development is adopted.

Fig. 4 Experimental approach

5.2. Data Generation

Once the algorithms are selected, the next step is to prepare a large volume of data for the experimental algorithms. This involves generating datasets with varying sizes and complexities to simulate real-world scenarios. The generated data represents individuals affected by COVID-19 from various towns, small cities, and villages. It's essential to sort this data to identify the areas with the least and the greatest number of affected individuals. The datasets include a wide range of numerical values, including duplicates, to assess the algorithms' performance under different conditions. By generating diverse datasets, the experiment aims to capture the algorithms' energy consumption across a spectrum of input scenarios, enabling comprehensive analysis and comparison.

5.3. Power Measurement

To track energy consumption during the execution process, a Power Reading Unit (PRU) measurement module is utilized. This module allows for accurate measurement of energy consumption at hardware runtime, providing insights into the algorithms' energy usage patterns. By measuring power consumption in real time, the experiment captures precise data on energy consumption, enabling meaningful analysis and evaluation of algorithmic efficiency.

5.4. Energy Consumption Analysis

The collected data on power consumption is now ready for the next phase, which involves analysing the algorithms to determine the most energy-consuming parts. The in-depth analysis examines the structure, execution flow, and resource utilization of each algorithm to identify sections that are energy-intensive. By pinpointing and evaluating these areas, the experiment reveals potential opportunities for optimizing energy use and improving the efficiency of the algorithms. This analysis forms the basis for informed decision-making in software development, steering the focus towards the optimization of code for energy efficiency.

5.5. Energy Usage Report Generation

The final step entails generating a comprehensive report on energy usage that focuses on green metrics. This report synthesizes the findings from the energy consumption analysis, highlighting key insights, trends, and recommendations for improving energy efficiency in software development. The report serves as a valuable resource for stakeholders, providing actionable insights and guidance for promoting sustainability in the tech industry. By emphasizing green metrics, the report underscores the importance of energy efficiency in software development and advocates for environmentally conscious practices. Overall, this approach offers a systematic and structured framework for analysing the energy efficiency of sorting algorithms, contributing to the broader goal of sustainability in the tech industry. Through rigorous experimentation, data-driven analysis, and informed decision-making, the approach facilitates the development of energy-efficient software solutions, aligning with principles of environmental stewardship and responsible technological innovation.

6. Experimental Design and validation

The experimental design and validation were meticulously designed and executed to assess the energy efficiency of both the sorting algorithms, Quick Sort and Bubble Sort, implemented in C#. Below is a detailed breakdown of each step:

6.1. Application Selection

For this experiment, C# for Windows and Python for Linux were chosen as the programming language due to its widespread usage and compatibility. The project code was developed within Visual Studio and Visual Code to ensure consistency and ease of execution.

6.2. Data Generation

To conduct a comprehensive analysis, a dataset comprising approximately 100 thousand numerical values ranging from single to six-digit integers was generated. This dataset included duplicate values to assess the sorting algorithms' performance under varying conditions and data complexities.

6.3. Power Measurement

Energy consumption during the execution process was measured using a Power Reading Unit (PRU) integrated into the experimental computer. Prior to executing the sorting algorithms, the computer's idle time energy consumption was recorded to establish a baseline. The computer's specifications, including processor, RAM, and operating system details, were documented to provide context for the experiment. The configuration of the computer is as below: The system is powered by an Intel Core i3-3220 CPU processor, clocking at 3.30 GHz. It is equipped with 8.00 GB of RAM. The machine dual-boots Windows 10 Pro (Version 22H2) and Ubuntu 22.04 Linux. It's a 64-bit operating system running on an x64-based processor. The build number for the Windows OS is 19045.3930.

6.4. Energy Consumption Analysis

In this phase, C# & Python implementations of both Bubble Sort and Quick Sort algorithms were developed within the Visual Studio Code. The algorithms were structured as console applications to facilitate straightforward execution and output viewing in both Operating Systems like Windows and Linux.

6.4.1. Implementing Bubble Sort in C#: An Energy Efficiency Perspective

This Python (Figure 5) and C# (Figure 6) code executes the Bubble Sort technique on a set of integers and concurrently calculates the duration of the sorting operation. The `sampledata` function is defined, which returns an array of integers. In this case, the array is a 100 thousand integer dataset. The bubble_sort function is crafted to arrange an integer array in an increasing pattern, leveraging the Bubble Sort algorithm. This algorithm's mechanism involves the continuous exchange of adjacent elements if they are improperly ordered. The start time of the array creation is printed using the Datetime function to get the current date and time in milliseconds. The sampledata function is called to get the array of integers. The start time of the Bubble Sort process is printed, similar to the start time of the array creation. The function named bubble_sort is invoked to arrange the elements of the array in order.

The end time of the Bubble Sort process is printed, again similar to the start time of the array creation. Finally, the sorted array is printed to the console. This code serves as a textbook illustration of the Bubble Sort algorithm. This uncomplicated sorting algorithm continuously traverses the list, juxtaposes neighbouring elements, and interchanges them if they are incorrectly ordered. The traversal of the list is reiterated until the list is sorted. The time complexity of Bubble Sort is $O(n^2)$ in both the worst and average scenarios, where 'n' represents the quantity of items being sorted. Bubble Sort is easy to comprehend and implement, whether you're a novice or a seasoned developer.

6.4.2. Implementing Quick Sort in C#: An Energy Efficiency Perspective

Similarly, the below Python (Figure 7) and C# (Figure 8) code implements the Quick Sort algorithm on an array of integers and also measures the time taken for the sorting process. The `sampledata` function is defined, which returns an array of integers. In this case, the array is a 100 thousand integer dataset. The quick_sort function is defined to organize an integer array in ascending sequence using the Quick Sort algorithm. This algorithm operates by continuously interchanging adjacent elements if they are not in the correct sequence. The start time of the array creation is printed using the Datetime function to get the current date and time in milliseconds. The sampledata function is called to get the array of integers. The start time of the Quick Sort process is printed, similar to the start time of the array creation. The quick_sort function is called to sort the array. The end time of the Quick Sort process is printed, again similar to the start time of the array creation. Finally, the sorted array is printed to the console.

```
public static int[] sampledata() {…
}
public static void bubble_sort(int[] arr) {
    int temp;
    for (int j = 0; j <= arr.Length - 2; j++)
        for (int i = 0; i <= arr.Length - 2; i++)
            if (arr[i] > arr[i + 1])
            {
                temp = arr[i + 1]; arr[i + 1] = arr[i]; arr[i] = temp;
            }
}

Console.WriteLine("Array Started: " + DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss.fff"));
int[] arr = sampledata();
Console.WriteLine("Bubble Sort Loop Started: " + DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss.fff"));
bubble_sort(arr);
Console.WriteLine("Sorted: " + DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss.fff"));
foreach (int p in arr)
    Console.Write(p + " ");
```

Fig. 5 Bubble sort algorithm C# code

```
import datetime
def sampledata():…

def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1] :
                arr[j], arr[j+1] = arr[j+1], arr[j]


print("Array Started: ", datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S.%f"))
arr = sampledata()
print("Bubble Sort Loop Started: ", datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S.%f"))
bubble_sort(arr)
print("Sorted: ", datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S.%f"))
print("Sorted array is: ", arr)
```

Fig. 6 Bubble sort algorithm python code

```
public static List<int> sampledata() {…
}

public static List<int> quick_sort(List<int> arr){
    if (arr.Count <= 1)
        return arr;
    int pivot = arr[arr.Count / 2];
    List<int> left = arr.Where(x => x < pivot).ToList();
    List<int> middle = arr.Where(x => x == pivot).ToList();
    List<int> right = arr.Where(x => x > pivot).ToList();
    return quick_sort(left).Concat(middle).Concat(quick_sort(right)).ToList();
}

Console.WriteLine("Array Started: " + DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss.fff"));
List<int> arr = sampledata();
Console.WriteLine("Quick Sort Loop Started: " + DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss.fff"));
quick_sort(arr);
Console.WriteLine("Sorted: " + DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss.fff"));
foreach (int p in arr)
    Console.Write(p + " ");
```

Fig. 7 Quick sort algorithm C# code

```
import datetime
def sampledata():…

def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quick_sort(left) + middle + quick_sort(right)


print("Array Started: ", datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S.%f"))
arr = sampledata()
print("Quick Sort Loop Started: ", datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S.%f"))
arr = quick_sort(arr)
print("Sorted: ", datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S.%f"))
print("Sorted array is: ", arr)
```

Fig. 8 Quick sort algorithm python code

```
SortingExample.exe
Array Started: 2024-01-27 00:01:09.723
Bubble Sort Loop Started: 2024-01-27 00:01:09.864
Sorted: 2024-01-27 00:02:27.386
3 3 28 50 60 69 74 88 102 112 126 146 147 153 165 165 179 179 182 185 192 198 206 213 214 229 233 239 254 273 287 300 309
 309 312 317 321 324 329 331 338 346 361 363 374 382 389 390 418 446 448 455 458 462 467 467 472 475 481 485 495 503 529
538 542 544 548 569 583 585 609 613 618 623 633 638 659 673 675 687 688 692 704 705 716 727 748 753 767 795 801 804 821 8
24 832 837 841 856 856 871 880 887 889 893 940 941 947 965 984 984 986 1001 1008 1019 1022 1024 1027 1029 1046 1051 1056
1061 1063 1066 1074 1080 1082 1091 1095 1108 1112 1132 1133 1135 1151 1168 1170 1177 1184 1204 1226 1228 1237 1255 1261 1
284 1301 1314 1334 1340 1342 1343 1358 1401 1409 1413 1420 1429 1431 1445 1448 1461 1470 1471 1474 1474 1485 1489 1500 15
17 1519 1527 1546 1550 1554 1571 1624 1635 1641 1672 1691 1702 1702 1709 1723 1732 1737 1745 1754 1767 1775 1795 1796 180
8 1813 1819 1822 1833 1846 1847 1886 1887 1892 1896 1897 1903 1907 1908 1914 1925 1927 1942 1942 1951 1966 1976 1984 1986
 2004 2007 2008 2025 2027 2027 2034 2039 2071 2092 2096 2099 2108 2122 2132 2134 2148 2162 2175 2178 2188 2194 2202 2245
2253 2275 2279 2295 2296 2302 2313 2316 2317 2318 2331 2334 2347 2362 2378 2396 2410 2421 2428 2431 2472 2475 2482 2485 2
488 2540 2551 2594 2604 2629 2630 2630 2638 2640 2655 2657 2661 2662 2662 2689 2698 2706 2723 2731 2752 2755 2775 2777 27
89 2794 2795 2796 2804 2809 2809 2810 2836 2836 2845 2853 2856 2861 2933 2948 2958 2971 2984 2986 2996 3019 3042 3052 306
0 3072 3076 3114 3114 3117 3120 3120 3123 3138 3139 3156 3179 3189 3193 3207 3208 3209 3213 3215 3219 3223 3237 3240 3245
 3246 3256 3257 3260 3266 3267 3275 3278 3297 3315 3335 3357 3364 3365 3373 3375 3388 3406 3410 3418 3425 3436 3445 3452
3459 3464 3469 3471 3489 3494 3494 3504 3508 3529 3545 3549 3555 3562 3565 3565 3570 3606 3632 3641 3646 3654 3666 3684 3
687 3690 3713 3724 3726 3730 3731 3743 3745 3745 3756 3783 3784 3790 3793 3798 3815 3823 3833 3842 3847 3849 3874 3878 38
83 3883 3887 3894 3917 3918 3919 3963 3980 4004 4032 4034 4039 4043 4058 4067 4067 4092 4109 4117 4120 4121 4140 4145 415
7 4160 4161 4162 4162 4163 4163 4176 4192 4202 4206 4212 4213 4220 4231 4257 4264 4270 4281 4282 4283 4299 4309 4311 4320
 4329 4338 4343 4343 4354 4359 4380 4384 4385 4402 4408 4415 4422 4423 4429 4446 4455 4461 4470 4471 4493 4494 4495 4503
4508 4534 4538 4540 4542 4565 4583 4584 4590 4595 4611 4613 4621 4627 4645 4648 4654 4656 4663 4663 4666 4698 4706 4723 4
730 4742 4748 4766 4768 4770 4775 4784 4804 4817 4844 4867 4881 4890 4895 4911 4912 4936 4950 4952 4962 4979 4980 4982 50
12 5024 5028 5028 5032 5035 5042 5043 5053 5066 5069 5100 5108 5108 5113 5122 5129 5130 5132 5136 5143 5154 5154 5175 517
7 5178 5188 5189 5207 5215 5220 5243 5246 5277 5284 5296 5314 5324 5338 5341 5352 5352 5395 5421 5432 5451 5470 5470 5477
 5478 5517 5521 5527 5532 5535 5537 5563 5565 5567 5573 5592 5599 5602 5603 5606 5609 5611 5656 5659 5675 5681 5683 5689
5701 5703 5729 5744 5795 5798 5811 5816 5830 5836 5838 5846 5852 5854 5862 5881 5910 5912 5937 5951 5960 5963 5971 5976 5
990 5992 6011 6022 6027 6060 6061 6074 6075 6081 6088 6100 6117 6124 6132 6138 6141 6176 6177 6188 6202 6207 6217 6220 62
33 6251 6259 6260 6260 6262 6302 6319 6323 6326 6334 6343 6352 6354 6362 6417 6424 6428 6437 6449 6453 6484 6489 6494 650
```

Fig. 9 Bubble sort sample code execution output

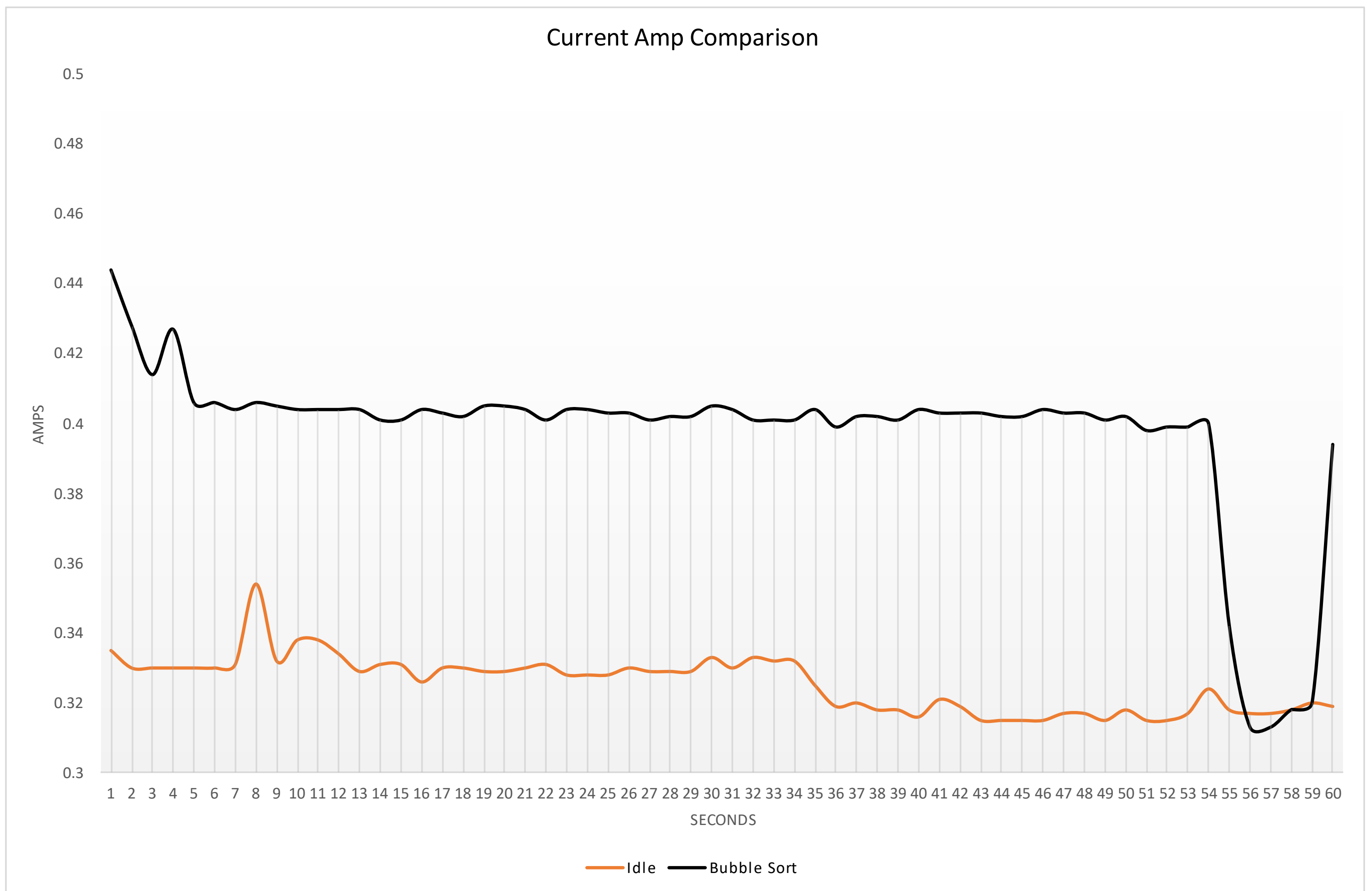

Fig. 10 Power (Amp) consumption comparison graph for bubble sorting and idle time of a computer

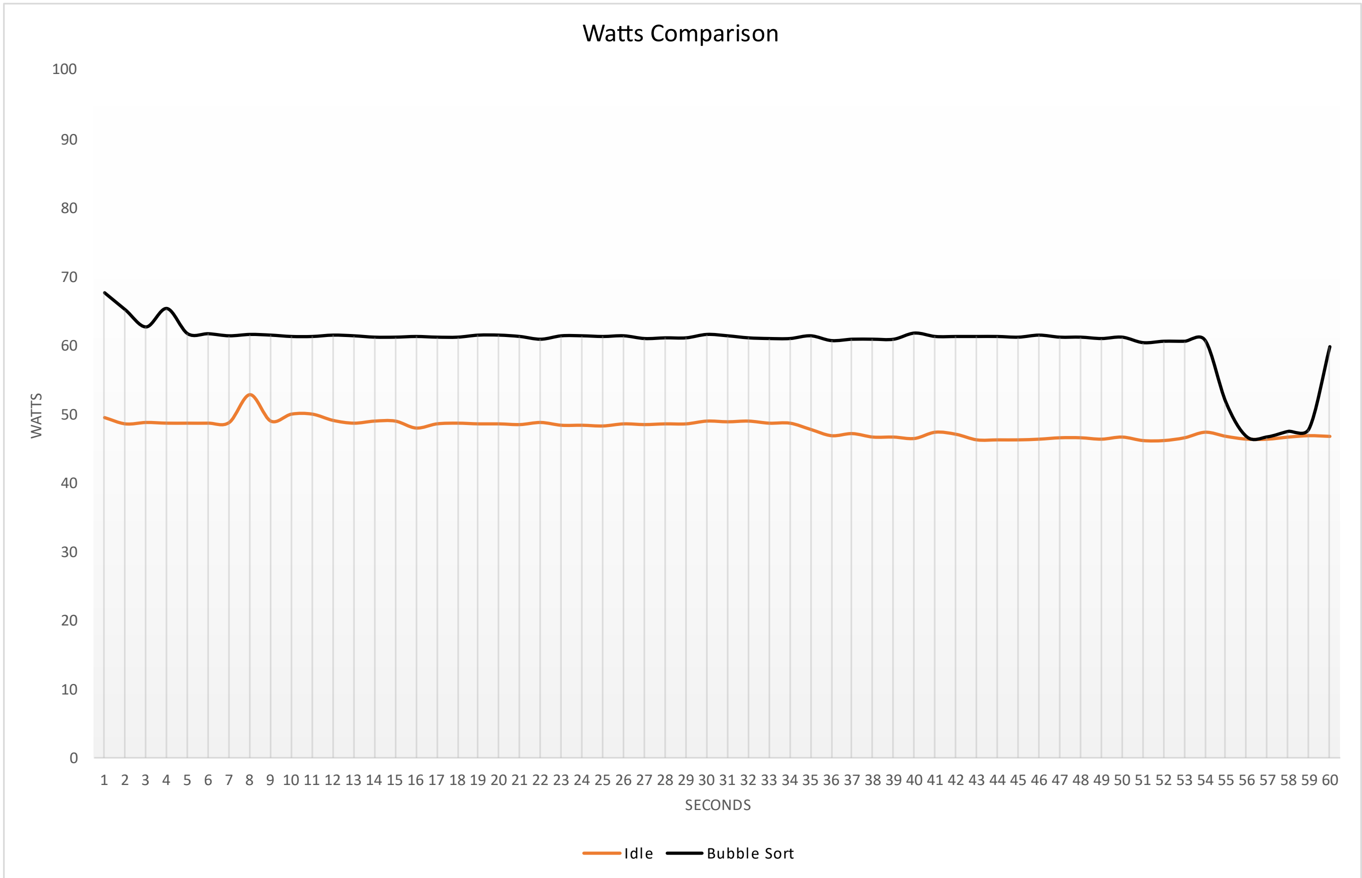

Fig. 11 Watts consumption comparison graph for Bubble sorting and idle time of a computer

```
C:\Users\easyf\source\repos\SortingExample\SortingExample\bin\Debug\net6.0\SortingExample.exe
Array Started: 2024-01-26 23:59:56.390
Quick Sort Loop Started: 2024-01-26 23:59:56.447
Sorted: 2024-01-26 23:59:56.535
3 3 28 50 60 69 74 88 102 112 126 146 147 153 165 165 179 179 182 185 192 198 206 213 214 229 233 239 254 273 287 300 309
 309 312 317 321 324 329 331 338 346 361 363 374 382 389 390 418 446 448 455 458 462 467 467 472 475 481 485 495 503 529
538 542 544 548 569 583 585 609 613 618 623 633 638 659 673 675 687 688 692 704 705 716 727 748 753 767 795 801 804 821 8
24 832 837 841 856 856 871 880 887 889 893 940 941 947 965 984 984 986 1001 1008 1019 1022 1024 1027 1029 1046 1051 1056
1061 1063 1066 1074 1080 1082 1091 1095 1108 1112 1132 1133 1135 1151 1168 1170 1177 1184 1204 1226 1228 1237 1255 1261 1
284 1301 1314 1334 1340 1342 1343 1358 1401 1409 1413 1420 1429 1431 1445 1448 1461 1470 1471 1474 1474 1485 1489 1500 15
17 1519 1527 1546 1550 1554 1571 1624 1635 1641 1672 1691 1702 1702 1709 1723 1732 1737 1745 1754 1767 1775 1795 1796 180
8 1813 1819 1822 1833 1846 1847 1886 1887 1892 1896 1897 1903 1907 1908 1914 1925 1927 1942 1942 1951 1966 1976 1984 1986
 2004 2007 2008 2025 2027 2027 2034 2039 2071 2092 2096 2099 2108 2122 2132 2134 2148 2162 2175 2178 2188 2194 2202 2245
2253 2275 2279 2295 2296 2302 2313 2316 2317 2318 2331 2334 2347 2362 2378 2396 2410 2421 2428 2431 2472 2475 2482 2485 2
488 2540 2551 2594 2604 2629 2630 2630 2638 2640 2655 2657 2661 2662 2662 2689 2698 2706 2723 2731 2752 2755 2775 2777 27
89 2794 2795 2796 2804 2809 2809 2810 2836 2836 2845 2853 2856 2861 2933 2948 2958 2971 2984 2986 2996 3019 3042 3052 306
0 3072 3076 3114 3114 3117 3120 3120 3123 3138 3139 3156 3179 3189 3193 3207 3208 3209 3213 3215 3219 3223 3237 3240 3245
 3246 3256 3257 3260 3266 3267 3275 3278 3297 3315 3335 3357 3364 3365 3373 3375 3388 3406 3410 3418 3425 3436 3445 3452
3459 3464 3469 3471 3489 3494 3494 3504 3508 3529 3545 3549 3555 3562 3565 3565 3570 3606 3632 3641 3646 3654 3666 3684 3
687 3690 3713 3724 3726 3730 3731 3743 3745 3745 3756 3783 3784 3790 3793 3798 3815 3823 3833 3842 3847 3849 3874 3878 38
83 3883 3887 3894 3917 3918 3919 3963 3980 4004 4032 4034 4039 4043 4058 4067 4067 4092 4109 4117 4120 4121 4140 4145 415
7 4160 4161 4162 4162 4163 4163 4176 4192 4202 4206 4212 4213 4220 4231 4257 4264 4270 4281 4282 4283 4299 4309 4311 4320
 4329 4338 4343 4343 4354 4359 4380 4384 4385 4402 4408 4415 4422 4423 4429 4446 4455 4461 4470 4471 4493 4494 4495 4503
4508 4534 4538 4540 4542 4565 4583 4584 4590 4595 4611 4613 4621 4627 4645 4648 4654 4656 4663 4663 4666 4698 4706 4723 4
730 4742 4748 4766 4768 4770 4775 4784 4804 4817 4844 4867 4881 4890 4895 4911 4912 4936 4950 4952 4962 4979 4980 4982 50
12 5024 5028 5028 5032 5035 5042 5043 5053 5066 5069 5100 5108 5108 5113 5122 5129 5130 5132 5136 5143 5154 5154 5175 517
7 5178 5188 5189 5207 5215 5220 5243 5246 5277 5284 5296 5314 5324 5338 5341 5352 5352 5395 5421 5432 5451 5470 5470 5477
 5478 5517 5521 5527 5532 5535 5537 5563 5565 5567 5573 5592 5599 5602 5603 5606 5609 5611 5656 5659 5675 5681 5683 5689
5701 5703 5729 5744 5795 5798 5811 5816 5830 5836 5838 5846 5852 5854 5862 5881 5910 5912 5937 5951 5960 5963 5971 5976 5
990 5992 6011 6022 6027 6060 6061 6074 6075 6081 6088 6100 6117 6124 6132 6138 6141 6176 6177 6188 6202 6207 6217 6220 62
33 6251 6259 6260 6260 6262 6302 6319 6323 6326 6334 6343 6352 6354 6362 6417 6424 6428 6437 6449 6453 6484 6489 6494 650
```

Fig. 12 Quick Sort sample code execution output

This code serves as a textbook illustration of the Quick Sort algorithm. This efficient, in-place sorting algorithm partitions a large array into two smaller sub-arrays (the elements that are less and the greater elements) and recursively arranges them. The time complexity of Quick Sort is O(n log n) under optimal and average conditions and O(n2) under the least favourable condition, where ‘n’ denotes the number of elements being sorted. After code preparation, execution was carried out (Figures 9, 12), and results were captured for analysis and comparison.

7. Result

The examination of a Bubble Sort algorithm initiates with the acquisition of data from the Power Reading Unit (PRU), a crucial setup for precisely gauging power consumption during algorithm execution. The initial step involves recording the computer's idle time for one minute, serving as a baseline measurement for power consumption when no significant processes are active. Subsequent to idle time recording, the Bubble Sort program is executed, and power consumption data is recorded from the program's initiation until completion. Simultaneously, Quick Sort executes within a fraction of a second, resulting in null power consumption for that period. After recording both sets of power data—idle time consumption and Bubble Sort execution consumption—they are compared and analysed to discern changes in power consumption during algorithmic execution.

Table 1. Power (Amp) consumption comparison table for Bubble sorting and idle time of a computer

Seconds	Idle	Bubble Sort
1	0.335	0.444
2	0.330	0.428
3	0.330	0.414
4	0.330	0.427
5	0.330	0.406
6	0.330	0.406
7	0.331	0.404
8	0.354	0.406
9	0.332	0.405
10	0.338	0.404
11	0.338	0.404
12	0.334	0.404
13	0.329	0.404
14	0.331	0.401
15	0.331	0.401
16	0.326	0.404
17	0.330	0.403
18	0.330	0.402
19	0.329	0.405
20	0.329	0.405

Table 2. Watts consumption comparison table for Bubble sorting and idle time of a computer

Seconds	Idle	Bubble Sort
1	49.6	67.8
2	48.7	65.3
3	48.9	62.8
4	48.8	65.5
5	48.8	61.8
6	48.8	61.8
7	48.9	61.5
8	52.9	61.7
9	49.1	61.6
10	50.1	61.4
11	50.1	61.4
12	49.2	61.6
13	48.8	61.5
14	49.1	61.3
15	49.1	61.3
16	48.1	61.4
17	48.7	61.3
18	48.8	61.3
19	48.7	61.6
20	48.7	61.6

To enhance comprehension of the comparison, the data is converted into a graphical format (Figures 10, 11) and table format (Tables 1, 2), visually depicting the disparity in power consumption between the idle and active states during Bubble Sort execution. As Quick Sort executes within a second, data for Quick Sort is null. The analysis indicates that program execution, including the time-consuming Bubble Sort algorithm, induces an increase in power consumption.

This emphasizes the significance of efficient algorithm utilization. To mitigate high current consumption, it is advisable to optimize algorithm usage and explore alternative sorting methods when dealing with resource-intensive tasks, such as sorting large datasets.

A comparison has also been made of the lines of code between C# and Python. The details of this comparison are provided in Table 3. To calculate the energy consumed (in watt-hours or Wh), use the below formula.

$$E = P \times t$$

In this above formula:

- `E` represents the energy consumption measured in watt-hours (Wh),
- `P` denotes the power, quantified in watts (W),
- `t` signifies the duration in hours.

Table 3. Line of code comparison between C# & Python for Sorting algorithm

Sort Algorithm	C# Line of Code	Python Line of Code	Difference
Bubble Sort	16 Lines	15 Lines	1 Line
Quick Sort	18 Lines	17 Lines	1 Line

Table 4. Experiment execution timetable of sorting algorithms

Sort Algorithm	Start Time	End Time	Time Taken	Average Amps	Average Watts	Current Consumed (Wh)Watts x Hrs
Quick Sort Algorithm	23:59:56.447	23:59:56.535	00:00:00.088	0.32	48	48 x 0 = **0Wh**
Bubble Sort Algorithm	00:01:09.864	00:02:27.386	00:01:17.522	0.40	59	59 x 0.02 = **1.18Wh**

Table 4 details the execution time, Amps & Watts taken for both the sorting algorithm to process 100 thousand numerical data. Upon analyzing various parameters outlined in the previous tables (Tables 4 & 3), it's clear that the number of lines of code doesn't significantly influence the choice of programming language. It's noteworthy that the Quick Sort algorithm is highly energy-efficient, consuming almost no energy. On the other hand, the Bubble Sort algorithm consumes 1.18Wh, which represents a 100% increase when processing a large amount of input data. Diverging from conventional power efficiency experiments that utilize internal software to track power consumption metrics across CPU, memory, and GPU, this experiment employs a unique approach. It uses a custom-designed PRU, an embedded hardware device, to precisely measure voltage and wattage, thus providing accurate power consumption data during the execution of specific programs. In the realm of software development, the built-in PRU offers developers real-time power usage insights through diagnostic data, assisting them in enhancing or selecting the most efficient algorithms or solutions for distinct challenges.

8. Conclusion and Future Work

In conclusion, the study throws light on the critical role of energy efficiency in software development, particularly within the framework of Green Software development. By conducting a comparative analysis of Bubble Sort and Quick Sort algorithms, the experiment elucidates the energy consumption dynamics inherent in sorting large numerical datasets.

The findings reveal that the Quick Sort algorithm exhibits superior energy efficiency compared to Bubble Sort when processing extensive data inputs. Through meticulous experimentation and analysis, the study underscores the significance of adopting energy-efficient practices in software development, thereby contributing to the broader goal of environmental sustainability. The comprehensive methodology outlined in the study provides a structured framework for assessing and optimizing energy usage in software systems, offering valuable insights for stakeholders in the tech industry. Moving forward, efforts to integrate energy efficiency considerations into software development processes are paramount, ensuring the development of sustainable and environmentally conscious software solutions. As part of future work, there is a planned expansion of the current research to encompass an exploration of the energy efficiency profiles of additional commonly utilized tools and applications prevalent in everyday use. This extended investigation aims to broaden the scope of understanding regarding energy consumption patterns across a diverse range of software applications integral to daily activities. By delving into the energy efficiency metrics of these widely employed tools, insights can be gained into potential areas for optimization and enhancement, thereby facilitating the development of more energy-conscious and environmentally sustainable software solutions. Through systematic experimentation and analysis, future work to contribute to the ongoing advancement of Green Software development practices, fostering a culture of energy efficiency and environmental stewardship within the technology industry.

References

[1] Ariful Islam Shiplu, Mostafizer Rahman, and Yutaka Watanobe, "LSA: A Novel State-Of-The-Art Sorting Algorithm for Efficient Arrangement of Large Data," *Proceedings of the 2023 4th Asia Service Sciences and Software Engineering Conference*, pp. 105-111, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[2] Martin Mahaux, and Caroline Canon, "Integrating the Complexity of Sustainability in Requirements Engineering," *Proceedings of the 1st International Conference on Requirements Engineering for Sustainable Systems*, 2012. [Google Scholar] [Publisher Link]

[3] Christoph Becker et al., "Requirements: The Key to Sustainability," *IEEE Software*, vol. 33, no. 1, pp. 56-65, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[4] Eva Kern, Achim Guldner, and Stefan Naumann, *Including Software Aspects in Green IT: How to Create Awareness for Green Software Issues*, Green IT Engineering: Social, Business and Industrial Applications, Studies in Systems, Decision and Control, vol. 171, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[5] Stefanos Georgiou, Stamatia Rizou, and Diomidis Spinellis, "Software Development Lifecycle for Energy Efficiency: Techniques and Tools," *ACM Computing Surveys*, vol. 52, no. 4, pp. 1-33, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[6] Hayri Acar, "*Software Development Methodology in a Green IT Environment*," University of Lyon, pp. 1-121, 2017. [Google Scholar] [Publisher Link]

[7] Kerstin Eder, and John P. Gallagher, *Energy-Aware Software Engineering*, ICT - Energy Concepts for Energy Efficiency and Sustainability, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[8] Sara S. Mahmoud, and Imtiaz Ahmad, "A Green Model for Sustainable Software Engineering," *International Journal of Software Engineering and its Applications*, vol. 7, no. 4, pp. 1-20, 2013. [Google Scholar] [Publisher Link]

[9] Shantanu Ray et al., "Green Software Engineering Process : Moving Towards Sustainable Software Product Design," *Journal of Global Research in Computer Sciences*, vol. 4, no. 1, pp. 1-5, 2013. [Google Scholar] [Publisher Link]

[10] David Lo, "Human-Centered AI for Software Engineering: Requirements, Reflection, and Road Ahead," *Proceedings of the 16th Innovations in Software Engineering Conference*, Allahabad, India, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[11] Stefan Naumann et al., "The Greensoft Model: A Reference Model for Green and Sustainable Software and its Engineering," *Sustainable Computing: Informatics and Systems*, vol. 1, no. 4, pp. 294-304, 2011. [CrossRef] [Google Scholar] [Publisher Link]

[12] Amin Khalifeh et al., "Incorporating Sustainability Into Software Projects: A Conceptual Framework," *International Journal of Managing Projects in Business*, vol. 13, no. 6, pp. 1339-1361, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[13] Tribid Debbarma, and K. Chandrasekaran, "Green Measurement Metrics towards a Sustainable Software: A Systematic Literature Review," *2016 International Conference on Recent Advances and Innovations in Engineering (ICRAIE)*, Jaipur, India, pp. 1-7, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[14] Markus Dick et al., "Green Software Engineering with Agile Methods," *2013 2nd International Workshop on Green and Sustainable Software (GREENS)*, San Francisco, CA, USA, pp. 78-85, 2013. [CrossRef] [Google Scholar] [Publisher Link]