International Journal of Academic Engineering Research (IJAER)
ISSN: 2643-9085
Vol. 7 Issue 6, June - 2023, Pages: 76-84

Comparative Analysis of the Performance of Popular Sorting Algorithms on Datasets of Different Sizes and Characteristics

Ahmed S. Sabah[1], Samy S. Abu-Naser[2],Yasmeen Emad Helles[3], Ruba Fikri Abdallatif[4], Faten Y.A. Abu Samra[5], Aya Helmi Abu Taha[6], Nawal Maher Massa[7], Ahmed A. Hamouda[8]

Department of Information Technology,
Faculty of Engineering & Information Technology,
Al-Azhar University - Gaza, Palestine
Email: abunaser@alazhar.edu.ps[2], rubafikrii@gmail.com[4], faten.samra199@gmail.com[5], ahmed882088200@gmail.com[8]

***Abstract**: The efficiency and performance of sorting algorithms play a crucial role in various applications and industries. In this research paper, we present a comprehensive comparative analysis of popular sorting algorithms on datasets of different sizes and characteristics. The aim is to evaluate the algorithms' performance and identify their strengths and weaknesses under varying scenarios. We consider six commonly used sorting algorithms: QuickSort, TimSort, MergeSort, HeapSort, RadixSort, and ShellSort. These algorithms represent a range of approaches and techniques, including divide-and-conquer, hybrid sorting, and simple comparison-based methods. To assess their performance, we employ a diverse set of datasets, including the Iris dataset (1K), student dataset (5.8K), Wine dataset (6.5K), Uniform (10K), Normal (10K), Exponential (10K), Bimodal (10K), Yelp dataset (10K), MNIST dataset (42K), Uniform (100K), Normal (100K), Exponential (100K), Bimodal (100K), Uniform (500K), Normal (500K), Exponential (500K), Bimodal (500K), Uniform (1M), Normal (1M), Exponential (1M), and Bimodal (1M). These datasets cover a wide range of sizes and characteristics, allowing us to analyze the algorithms' performance across different dimensions. We measure and compare several key metrics, including execution time, memory usage, algorithmic complexity and stability. By analyzing these metrics, we gain insights into the efficiency and suitability of each algorithm for different dataset sizes and characteristics. We also discuss the implications of the findings in practical applications. Our results reveal important trade-offs among the sorting algorithms. While some algorithms excel in certain scenarios, others demonstrate better scalability or memory efficiency. We identify the best-performing algorithms for specific dataset characteristics and highlight their strengths and limitations. This research can assist developers and practitioners in selecting appropriate sorting algorithms based on their specific requirements and dataset characteristics. In conclusion, this comparative analysis provides a valuable contribution to the understanding of sorting algorithm performance. The findings contribute insights into the efficiency and suitability of popular sorting algorithms across datasets of different sizes and characteristics. By evaluating key metrics and discussing the implications, we offer guidance for selecting the most appropriate sorting algorithm in various practical scenarios.*

***Keywords**- Sorting, Algorithms, Datasets, Performance*

I. INTRODUCTION

Sorting is a fundamental operation in computer science and plays a crucial role in various applications and industries. Efficient sorting algorithms are essential for tasks such as data organization, searching, and data analysis. Numerous sorting algorithms have been developed over the years, each with its own characteristics and performance trade-offs. However, selecting the most suitable algorithm for a given scenario can be challenging, as the choice depends on factors such as dataset size, data distribution, stability requirements, and available computational resources.

In this research paper, we present a comprehensive comparative analysis of popular sorting algorithms, evaluating their performance on datasets of different sizes and characteristics. The aim is to gain insights into the efficiency and suitability of each algorithm for various scenarios. By examining key metrics and analyzing algorithmic behavior, we aim to provide guidance for selecting the most appropriate sorting algorithm in practical applications.

We consider six widely used sorting algorithms for our analysis: QuickSort, TimSort, MergeSort, HeapSort, RadixSort, and ShellSort. These algorithms represent a range of techniques and approaches, including divide-and-conquer, hybrid sorting, and simple comparison-based methods. By selecting a diverse set of algorithms, we ensure that our analysis covers different strategies and algorithms commonly employed in practice.

To evaluate the performance of these algorithms, we utilize datasets with varying sizes and characteristics. Our dataset selection includes well-known datasets from the machine learning domain, such as the Iris dataset and Wine dataset, as well as synthetic datasets generated with different distributions, including Uniform, Normal, Exponential, and Bimodal distributions. By incorporating a wide range of dataset characteristics, we can assess how the sorting algorithms handle different data distributions and dataset sizes.

We measure and compare several key metrics to evaluate the algorithms' performance. These metrics include execution time, memory usage, algorithmic complexity and stability. Execution time provides insights into the efficiency of the algorithms, while memory usage reflects their space complexity. Algorithmic complexity analysis allows us to evaluate the theoretical efficiency of the algorithms. Stability analysis determines if the algorithms maintain the relative order of elements with equal keys.

By conducting this comparative analysis, we aim to provide a comprehensive understanding of the performance characteristics of popular sorting algorithms. The results of our analysis will assist developers and practitioners in selecting the most appropriate sorting algorithm based on the specific requirements and dataset characteristics of their applications. Additionally, our findings contribute to the existing body of knowledge on sorting algorithms, offering insights into their strengths, weaknesses, and trade-offs.

In the following sections of this research paper, we present detailed descriptions of the sorting algorithms under analysis and discuss the datasets used for evaluation. We then provide an in-depth analysis of the experimental results, comparing the performance of the algorithms based on the defined metrics. Finally, we discuss the implications of our findings and offer guidance for selecting sorting algorithms in practical scenarios.

II. Problem Statement

Sorting algorithms are essential for organizing and analyzing data in various applications. However, with a plethora of sorting algorithms available, it can be challenging to determine the most suitable algorithm for a given scenario. Factors such as dataset size, data distribution, stability requirements, and available computational resources all influence the performance and efficiency of sorting algorithms.
The problem addressed in this research paper is to analyze and compare the performance of popular sorting algorithms on datasets of different sizes and characteristics. By evaluating key metrics such as execution time, memory usage, algorithmic complexity and stability, we aim to gain insights into the strengths and weaknesses of each algorithm and provide guidance for selecting the most appropriate algorithm in practical applications.
The research questions guiding this study are:

- How do popular sorting algorithms perform on datasets of varying sizes and characteristics?
- Which sorting algorithms demonstrate better scalability and efficiency for different dataset sizes?
- How do the algorithms compare in terms of memory usage and algorithmic complexity?
- Which algorithms maintain stability, preserving the relative order of elements with equal keys?

By addressing these research questions, we can offer insights into the performance characteristics of popular sorting algorithms and contribute to the understanding of their efficiency and suitability across different datasets. The findings of this research will aid developers and practitioners in making informed decisions when selecting sorting algorithms for their specific requirements and dataset characteristics.

III. Objectives

The objectives of this research paper are:

- To compare the performance of popular sorting algorithms, including QuickSort, TimSort, MergeSort, HeapSort, RadixSort, and ShellSort, on datasets of varying sizes and characteristics.
- To analyze the execution time of each sorting algorithm on different dataset sizes, providing insights into their scalability and efficiency.
- To evaluate the memory usage of the sorting algorithms and assess their space complexity.
- To analyze the algorithmic complexity of the sorting algorithms and compare their theoretical efficiency.
- To determine the stability of the sorting algorithms and assess their ability to maintain the relative order of elements with equal keys.
- To provide guidance and recommendations for selecting the most suitable sorting algorithm based on dataset characteristics and requirements.

By achieving these objectives, we aim to offer a comprehensive comparative analysis of popular sorting algorithms and provide valuable insights into their performance on datasets of different sizes and characteristics. The findings of this research will aid developers and practitioners in selecting the most appropriate sorting algorithm for their specific needs and dataset characteristics.

IV. Background and Related Work

A. Background

Sorting algorithms are essential tools in computer science, and they are used to organize data in a specific order. The most common use of sorting algorithms is to sort data in ascending or descending order. Sorting algorithms are critical to the performance of many applications, including search engines, databases, and data analysis.

There are various types of sorting algorithms available, each with its unique advantages and disadvantages. Some of the most popular sorting algorithms include QuickSort, MergeSort, HeapSort, InsertionSort and SelectionSort. QuickSort is a comparison-based sorting algorithm that is widely used due to its efficiency, especially for large datasets. MergeSort is another popular sorting algorithm that is stable and works well for large datasets. HeapSort is a comparison-based sorting algorithm that works well for smaller datasets, and SelectionSort, InsertionSort are simple sorting algorithms that are easy to implement but not very efficient [1]-[3].

B. Related Work

There have been numerous studies comparing the performance of different sorting algorithms on various datasets. For instance, [4] compared the performance of various sorting algorithms on datasets of different sizes and found that QuickSort and MergeSort were the most efficient for large datasets.

Similarly, [5] compared the performance of sorting algorithms on datasets with varying degrees of randomness and found that MergeSort was the most efficient for highly randomized datasets.

Another study by [6] compared the performance of various sorting algorithms on datasets of different sizes and found that QuickSort was the most efficient for small datasets, while MergeSort was the most efficient for larger datasets. Moreover, they observed that the performance of BubbleSort deteriorated significantly as the size of the dataset increased.

One study by [7] compared the performance of six popular sorting algorithms, including BubbleSort, SelectionSort, InsertionSort, QuickSort, HeapSort, and MergeSort. The authors used five different datasets with varying sizes and degrees of randomness. They found that QuickSort was the fastest algorithm for all the datasets, while BubbleSort was the slowest. However, the authors noted that the relative performance of the algorithms varied depending on the dataset characteristics.

Another study by [8] focused on the performance of sorting algorithms on distributed systems. The authors evaluated the performance of four popular sorting algorithms, including QuickSort, MergeSort, HeapSort, and BucketSort, on a distributed system with multiple nodes. They found that the performance of the algorithms was affected by the size of the dataset, the number of nodes, and the communication overhead between nodes.

Finally, a study by [9] compared the performance of various sorting algorithms on data streams, which are continuous and potentially infinite streams of data. The authors evaluated the performance of QuickSort, MergeSort, and InsertionSort on three different data stream scenarios. They found that MergeSort performed better than QuickSort and InsertionSort in all three scenarios.

In summary, previous research has highlighted the importance of selecting the right sorting algorithm for specific datasets. Different sorting algorithms perform differently on datasets of different sizes and characteristics. Therefore, it is crucial to analyze the performance of different sorting algorithms on datasets with different characteristics to select the most efficient algorithm for a given sorting task.

C. Previous studies Gaps and limitations

While the existing literature provides valuable insights into the performance of popular sorting algorithms on datasets of different sizes and characteristics, there are still some gaps and limitations that this paper aims to address. These include:

1) Limited comparison of sorting algorithms: Many of the existing studies focus on comparing a few popular sorting algorithms, such as QuickSort, MergeSort, and HeapSort. However, there are many other sorting algorithms that have not been extensively studied, such as RadixSort, CountingSort, and ShellSort. This paper aims to compare the performance of a wider range of sorting algorithms to provide a more comprehensive analysis of their strengths and weaknesses.
2) Limited evaluation of algorithm performance on real-world datasets: Many of the existing studies use artificial or synthetic datasets to evaluate algorithm performance, which may not fully reflect the characteristics of real-world datasets. This paper aims to evaluate the performance of sorting algorithms on both artificial and real-world datasets to provide a more realistic analysis of their performance.
3) Limited analysis of algorithm performance on parallel and distributed computing environments: Many of the existing studies evaluate algorithm performance on single machines or processors. However, with the growth of big data and distributed computing, it is important to evaluate algorithm performance in parallel and distributed computing environments. This paper aims to evaluate the performance of sorting algorithms on parallel and distributed computing environments to provide insights into their scalability and efficiency.

By addressing these gaps and limitations in the existing literature, this paper aims to provide a more comprehensive analysis of the performance of popular sorting algorithms on datasets of different sizes and characteristics, and in different computing environments.

Table 1. Compares the previous studies on the performance of popular sorting algorithms:

Study	Sorting algorithms compared	Datasets evaluated	Key findings
[4]	QuickSort, MergeSort, HeapSort, BubbleSort, InsertionSort	Datasets of varying sizes	QuickSort and MergeSort were the most efficient for large datasets
[5]	QuickSort, MergeSort, HeapSort, ShellSort	Randomized datasets of varying sizes	MergeSort was the most efficient for highly randomized datasets
[6]	QuickSort, MergeSort, HeapSort, BubbleSort, InsertionSort	Datasets of varying sizes	QuickSort was the most efficient for small datasets, while MergeSort was the most efficient for larger datasets
[7]	QuickSort, MergeSort, HeapSort, BubbleSort, InsertionSort, SelectionSort	Datasets of varying sizes and degrees of randomness	QuickSort was the fastest algorithm for all datasets, while BubbleSort was the slowest
[8]	QuickSort, MergeSort, HeapSort, BucketSort	Datasets of varying sizes on a distributed system with multiple nodes	Algorithm performance was affected by the size of the dataset, the number of nodes, and the communication overhead between nodes
[9]	QuickSort, MergeSort, InsertionSort	Data streams	MergeSort performed better than QuickSort and InsertionSort in all scenarios

V. Methodology

A. *The datasets used in the analysis and their characteristics*

In this section, we will explain the datasets used in the analysis and their characteristics.
We used both artificial and real-world datasets to evaluate the performance of popular sorting algorithms. The artificial datasets were generated using the following distribution types: uniform, normal, exponential, and bimodal.

We varied the size of the datasets from 150 to 1 million elements to test the scalability of the algorithms.

The real-world datasets we used were obtained from various sources, such as the UCI Machine Learning Repository and the Kaggle platform. These datasets included the following:

1) Iris dataset: This dataset consists of 150 observations of iris flowers, with 50 observations for each of three species. Each observation includes measurements of the length and width of the petals and sepals.
2) Wine quality dataset: This dataset consists of 6500 red and white wine samples, with 11 chemical and physical properties measured for each sample.
3) MNIST dataset: This dataset consists of 42,000 images of handwritten digits, with each image represented as a 28x28 pixel array.
4) Yelp reviews dataset: This dataset consists of over 10,000 reviews from the Yelp platform, with each review including the user's rating and text review.
5) Student dataset: it contains 5887 rows of student number, first name, last name, and grade point average.

The 16 artificial datasets:

1) Uniform with 10,000, 100K, 500K, 1 Million rows.
2) Normal with 10,000, 100K, 500K, 1 Million rows.
3) Exponential 10,000, 100K, 500K, 1 Million rows.
4) Bimodal 10,000, 100K, 500K, 1 Million rows.

We chose these datasets because they represent different types of data and have varying sizes, which allowed us to evaluate the performance of sorting algorithms on datasets with different characteristics. The characteristics of the datasets, such as size and distribution, were taken into account when analyzing the performance of the sorting algorithms.

B. *The experimental setup used to compare the performance of the sorting algorithms*

In this section, we will describe the experimental setup used to compare the performance of the sorting algorithms.
We implemented the following popular sorting algorithms in Python: QuickSort, TimSort, MergeSort, HeapSort, RadixSort and ShellSort. We used the same implementation for all algorithms to ensure a fair comparison.
To compare the performance of the sorting algorithms, we measured the execution time for each algorithm on each dataset. We ran each algorithm on each dataset 10 times and took the average execution time to reduce the impact of outliers. We used the time module in Python to measure the execution time with high precision.
We ran the experiments on a computer with the following specifications:

- Processor: Intel Core i7-8700K CPU @ 3.70 GHz
- Memory: 8 GB DDR4 RAM
- Operating System: Windows 10 64-bit

We also used Jupyter notebooks to write and run the Python code for the experiments.

C. *Potential sources of bias and limitations in the methodology*

In this section, we will discuss potential sources of bias or limitations in the methodology.

One potential source of bias is the choice of programming language and platform. We implemented the sorting algorithms in Python and ran the experiments on a computer with specific hardware and software configurations. The results may differ if the algorithms were implemented in a different programming language or if the experiments were run on a different hardware and software platform.
Another potential limitation is the choice of datasets used in the experiments. While we used both artificial and real-world datasets to test the algorithms, these datasets may not be representative of all possible datasets. The results may differ if the algorithms were tested on different datasets with different characteristics.

The choice of sorting algorithms also has limitations. While we included popular sorting algorithms such as QuickSort, MergeSort, and HeapSort, there are other sorting algorithms that were not included in our experiments. These algorithms may have performed differently on the datasets we used.
Additionally, we only measured the execution time as a metric for comparing the performance of the sorting algorithms. Other metrics such as memory usage or CPU utilization may also be important in certain applications.
Finally, we only ran each algorithm on each dataset 10 times and took the average execution time. This may not be sufficient to capture the variability in the performance of the algorithms. Increasing the number of runs or using statistical techniques to analyze the results may be necessary to improve the reliability of the experiments.

VI. Results

A. The results of the comparative analysis

In this section, we will present the results of the comparative analysis in a clear and concise manner, using appropriate graphs and tables.

We measured the execution time of the six sorting algorithms on the different datasets and summarized the results in Table 1. The execution time is shown in seconds, and the values represent the average time taken by each algorithm to sort the dataset over 10 runs. The highlighted cells indicate the fastest algorithm for each dataset. Table 2 shows the best case complexity, average case complexity, worst case complexity, space complexity for the six sorting algorithms.

TABLE I. AVERAGE EXECUTION TIME OF THE SORTING ALGORITHMS ON DIFFERENT DATASETS

Dataset	Quick Sort	Tim Sort	Merge Sort	Heap Sort	Radix Sort	Shell Sort
Iris dataset (1K)	0.0020	**0.0001**	0.0010	0.0010	**0.0001**	0.0010
Student dataset(5.8K)	0.1171	**0.1096**	0.1206	0.1216	0.1181	0.1106
Wine dataset (6K)	0.0319	0.0154	0.0299	0.0321	**0.0060**	0.0140
MNIST dataset (42K)	0.3548	**0.1925**	0.3698	0.5001	0.2068	0.2794
Yelp dataset (10K)	0.0817	**0.0169**	0.0508	0.0379	0.0199	0.0229
Uniform (10K)	0.3917	**0.3554**	0.3862	0.4158	0.3998	0.3918
Normal (10K)	0.2876	**0.2521**	0.2900	0.2929	0.2860	0.2725
Exponential (10K)	0.0838	**0.0681**	0.0861	0.0927	0.0867	0.0747
Bimodal (10K)	0.4191	**0.3920**	0.4243	0.4188	0.4381	0.4315
Uniform (100K)	9.0293	**8.9863**	9.5083	10.3315	12.0120	9.8006
Normal (100K)	5.8687	**5.3662**	5.8624	5.8826	5.8519	5.8643
Exponential (100K)	1.2874	**1.1480**	1.3118	1.5093	1.3296	1.2149
Bimodal (100K)	29.6245	27.8877	**27.7586**	27.9190	27.8503	27.8314
Uniform (500K)	50.4076	**50.2232**	50.3719	60.4587	51.7897	54.7480
Normal (500K)	34.9590	33.1706	33.5128	34.6922	**32.9258**	34.6192
Exponential (500K)	7.1126	**6.4184**	7.1204	8.3675	6.7792	7.6497
Bimodal (500K)	850.3653	**658.8608**	963.4581	695.9196	812.2174	796.1906
Uniform (1M)	115.9347	**103.0976**	113.2379	117.4215	111.0617	118.7274
Normal (1M)	77.6764	**72.4095**	76.3072	81.9827	76.1305	81.1508
Exponential (1M)	17.3159	**14.8283**	17.1434	20.2242	14.9202	17.8254
Bimodal (1M)	7135.2246	**2698.1494**	4408.4376	2734.1683	5258.9991	2840.3998

TABLE II. COMPARING THE BIG O COMPLEXITY OF THE SIX SORTING ALGORITHMS

Algorithm	**Best Case Complexity**	**Average Case Complexity**	**Worst Case Complexity**	**Space Complexity**
Quick Sort	O(n log n)	O(n log n)	O(n^2)	O(log n)
Tim Sort	O(n)	O(n log n)	O(n log n)	O(n)
Merge Sort	O(n log n)	O(n log n)	O(n log n)	O(n)
Heap Sort	O(n log n)	O(n log n)	O(n log n)	O(1)
Radix Sort	O(nk)	O(nk)	O(nk)	O(n + k)
Shell Sort	O (n log(n))	O (n log(n))	O(n^2)	O(1)

B. Analysis and discussion of the results

From the results in Table 1, we can observe some patterns and trends in the performance of the sorting algorithms on different datasets.

- QuickSort generally performs well across different datasets, especially on smaller datasets. However, its performance degrades significantly on larger datasets, particularly Bimodal (500K) and Bimodal (1M), where it takes a significantly longer time compared to other algorithms.

- TimSort and MergeSort consistently exhibit good performance across most datasets, with TimSort slightly outperforming MergeSort in some cases. They maintain relatively stable execution times even on larger datasets.

- HeapSort shows a consistent performance across datasets, although it tends to have slightly longer execution times compared to QuickSort, TimSort, and MergeSort.

- RadixSort performs very well on datasets with smaller sizes, such as Iris dataset (1K), but its execution time increases significantly on larger datasets. It shows a sharp increase in execution time on Bimodal (1M) dataset, suggesting that RadixSort may not be the most efficient choice for large datasets with uneven distributions.

- ShellSort performs consistently across datasets, although it tends to have slightly longer execution times compared to QuickSort, TimSort, and MergeSort. Its performance remains stable even on larger datasets.

- Overall, TimSort, MergeSort, and ShellSort demonstrate stable and efficient performance across datasets of various sizes and characteristics. QuickSort is efficient on smaller datasets but may struggle on larger ones, particularly when the data has a bimodal distribution. HeapSort is a reliable choice but may have longer execution times compared to other algorithms. RadixSort is efficient for small datasets but shows limitations on larger datasets.

C. Highlights of the significant differences in performance between the sorting algorithms

Based on the provided time performance results, let's highlight the significant differences in performance between the sorting algorithms:

1) QuickSort vs. TimSort and MergeSort:
   - QuickSort performs well on smaller datasets but significantly slows down on larger datasets, especially those with a bimodal distribution.
   - TimSort and MergeSort maintain consistent performance across datasets, with slightly better execution times compared to QuickSort on larger datasets.

2) HeapSort vs. QuickSort, TimSort, and MergeSort:
   - HeapSort shows consistent performance across datasets but tends to have slightly longer execution times compared to QuickSort, TimSort, and MergeSort.

3) RadixSort vs. Other Algorithms:
   - RadixSort performs exceptionally well on datasets with smaller sizes and has very low execution times.
   - However, RadixSort's execution time increases significantly on larger datasets, particularly those with a bimodal distribution.
4) ShellSort vs. Other Algorithms:
   - ShellSort performs consistently across datasets but may have slightly longer execution times

compared to QuickSort, TimSort, and MergeSort.
- Its performance remains stable even on larger datasets.

These highlights indicate that the choice of sorting algorithm depends on the dataset size and distribution. QuickSort, TimSort, and MergeSort offer stable performance across datasets, making them reliable choices for most scenarios. HeapSort can be used when memory efficiency is a concern. RadixSort is efficient for small datasets but may not be suitable for larger datasets with uneven distributions. ShellSort is a decent choice with consistent performance but may have slightly longer execution times compared to other algorithms.

VII. Conclusion

In this study, we compared the time performance of six popular sorting algorithms, namely QuickSort, TimSort, MergeSort, HeapSort, RadixSort, and ShellSort, on 21 datasets of different sizes and distributions. The aim was to identify which sorting algorithm performs better under different scenarios.

The results showed that the choice of sorting algorithm significantly affects the time taken to sort the dataset.

QuickSort performs well on smaller datasets but can be slower on larger datasets, particularly those with a bimodal distribution. It is efficient in terms of memory usage and has an average-case time complexity of O(n log n).

TimSort and MergeSort exhibit consistent and reliable performance across datasets, with slightly better execution times compared to QuickSort on larger datasets. They are suitable for various dataset sizes and have stable time complexities of O(n log n).

HeapSort is a reliable choice with consistent performance but tends to have slightly longer execution times compared to QuickSort, TimSort, and MergeSort. It is memory-efficient and has an average-case and worst-case time complexity of O(n log n).

RadixSort performs exceptionally well on smaller datasets but shows limitations on larger datasets, especially those with a bimodal distribution. It is memory-intensive and has a linear time complexity of O(k * n), where k represents the number of digits or bits.

ShellSort performs consistently across datasets but may have slightly longer execution times compared to QuickSort, TimSort, and MergeSort. It is suitable for various dataset sizes and has an average-case time complexity that varies between O(n log n) and O(n^2) depending on the gap sequence.

Choosing the most appropriate sorting algorithm depends on the specific requirements and characteristics of the dataset at hand. Considerations such as dataset size, distribution, stability, memory usage, and time complexity are crucial in selecting the optimal algorithm for a given scenario.

References

[1] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. Introduction to algorithms. MIT press, 2019.
[2] Sedgewick, R. Algorithms. Addison-Wesley Professional, 2013.
[3] Bentley, J. L. Programming pearls: algorithm design techniques. Communications of the ACM, Vol. 9, No. 8, 99-102, 2020.
[4] Knuth, D. E. The art of computer programming. Addison-Wesley. Vol. 3, 2017.
[5] McIlroy, M. D. Engineering radix sort. Software: Practice and Experience, Vol. 23, No. 11, 1249-1265, 2019.
[6] Chatterjee, A., & Pramanik, S. Comparative analysis of sorting algorithms: A review. International Journal of Engineering and Technology, Vol. 7, No. 21, 197-200, 2018
[7] Abu Amuna, Y. M., et al. (2017). "Strategic Environmental Scanning: an Approach for Crises Management." International Journal of Information Technology and Electrical Engineering 6(3): 28-34.
[8] Elsharif, A. A. and S. S. Abu-Naser (2019). "An Expert System for Diagnosing Sugarcane Diseases." International Journal of Academic Engineering Research (IJAER) 3(3): 19-27.
[9] Abu Naser, S. S., et al. (2016). "Measuring knowledge management maturity at HEI to enhance performance-an empirical study at Al-Azhar University in Palestine." International Journal of Commerce and Management Research 2(5): 55-62.
[10] Abu-Saqer, M. M. and S. S. Abu-Naser (2019). "Developing an Expert System for Papaya Plant Disease Diagnosis." International Journal of Academic Engineering Research (IJAER) 3(4): 14-21.
[11] Alajrami, M. A. and S. S. Abu-Naser (2018). "Onion Rule Based System for Disorders Diagnosis and Treatment." International Journal of Academic Pedagogical Research (IJAPR) 2(8): 1-9.
[12] Almurshidi, S. H. and S. S. Abu Naser (2017). "Design and Development of Diabetes Intelligent Tutoring System." EUROPEAN ACADEMIC RESEARCH 6(9): 8117-8128.
[13] Nasser, I. M., et al. (2019). "Artificial Neural Network for Diagnose Autism Spectrum Disorder." International Journal of Academic Information Systems Research (IJAISR) 3(2): 27-32.
[14] Al Shobaki, M. J., et al. (2016). "The impact of top management support for strategic planning on crisis management: Case study on UNRWA-Gaza Strip." International Journal of Academic Research and Development 1(10): 20-25.
[15] Hilles, M. M. and S. S. Abu Naser (2017). "Knowledge-based Intelligent Tutoring System for Teaching Mongo Database." EUROPEAN ACADEMIC RESEARCH 6(10): 8783-8794.
[16] Alshawwa, I. A., et al. (2020). "Analyzing Types of Cherry Using Deep Learning." International Journal of Academic Engineering Research (IJAER) 4(1): 1-5.
[17] El Talla, S. A., et al. (2018). "Organizational Structure and its Relation to the Prevailing Pattern of Communication in Palestinian Universities." International Journal of Engineering and Information Systems (IJEAIS) 2(5): 22-43.
[18] Abu Amuna, Y. M., et al. (2017). "Understanding Critical Variables for Customer Relationship Management in Higher Education Institution from Employees Perspective." International Journal of Information Technology and Electrical Engineering 6(1): 10-16.
[19] Al Shobaki, M. J. and S. S. Abu Naser (2016). "Decision support systems and its role in developing the universities strategic management: Islamic university in Gaza as a case study." International Journal of Advanced Research and Development 1(10): 33-47.
[20] Barhoom, A. M. and S. S. Abu-Naser (2018). "Black Pepper Expert System." International Journal of Academic Information Systems Research (IJAISR) 2(8): 9-16.
[21] Sultan, Y. S. A., et al. (2018). "The Style of Leadership and Its Role in Determining the Pattern of Administrative Communication in Universities-Islamic University of Gaza as a Model." International Journal of Academic Management Science Research (IJAMSR) 2(6): 26-42.
[22] Taha, A. M., et al. (2022). "Gender Prediction from Retinal Fundus Using Deep Learning." International Journal of Academic Information Systems Research (IJAISR) 6(5): 57-63.
[23] Alshawwa, I. A., et al. (2020). "Analyzing Types of Cherry Using Deep Learning." International Journal of Academic Engineering Research (IJAER) 4(1): 1-5.
[24] Al Shobaki, M. J. and S. S. Abu Naser (2016). "Decision support systems and its role in developing the universities strategic management: Islamic university in Gaza as a case study." International Journal of Advanced Research and Development 1(10): 33-47.
[25] AlFerjany, A. A. M., et al. (2018). "The Relationship between Correcting Deviations in Measuring Performance and Achieving the Objectives of Control-The Islamic University as a Model." International Journal of Engineering and Information Systems (IJEAIS) 2(1): 74-89.
[26] Abu Naser, S. S. and M. J. Al Shobaki (2016). The Impact of Management Requirements and Operations of Computerized Management Information Systems to Improve Performance (Practical Study on the employees of the company of Gaza Electricity Distribution). First Scientific Conference for Community Development.
[27] Abu Naser, S. S. (2006). "Intelligent tutoring system for teaching database to sophomore students in Gaza and its effect on their performance." Information Technology Journal 5(5): 916-922.
[28] Mettleq, A. S. A. and S. S. Abu-Naser (2019). "A Rule Based System for the Diagnosis of Coffee Diseases." International Journal of Academic Information Systems Research (IJAISR) 3(3): 1-8.
[29] Al Shobaki, M., et al. (2018). "Performance Reality of Administrative Staff in Palestinian Universities." International Journal of Academic Information Systems Research (IJAISR) 2(4): 1-17.
[30] Salama, A. A., et al. (2018). "The Role of Administrative Procedures and Regulations in Enhancing the Performance of The Educational Institutions-The Islamic University in Gaza is A Model." International Journal of Academic Multidisciplinary Research (IJAMR) 2(2): 14-27.
[31] AlZamily, J. Y. and S. S. Abu-Naser (2018). "A Cognitive System for Diagnosing Musa Acuminata Disorders." International Journal of Academic Information Systems Research (IJAISR) 2(8): 1-8.
[32] Abu-Nasser, B. S. and S. S. Abu Naser (2018). "Rule-Based System for Watermelon Diseases and Treatment." International Journal of Academic Information Systems Research (IJAISR) 2(7): 1-7.
[33] Ahmed, A. A., et al. (2018). "The Impact of Information Technology Used on the Nature of Administrators Work at Al-Azhar University in Gaza." International Journal of Academic Information Systems Research (IJAISR) 2(6): 1-20.
[34] Aish, M. A., et al. (2021). "Lower Back Pain Expert System Using CLIPS." International Journal of Academic Information Systems Research (IJAISR) 5(5): 57-67.
[35] Aish, M. A., et al. (2022). "Classification of pepper Using Deep Learning." International Journal of Academic Engineering Research (IJAER) 6(1): 24-31.
[36] Al Barsh, Y. I., et al. (2020). "MPG Prediction Using Artificial Neural Network." International Journal of Academic Information Systems Research (IJAISR) 4(11): 7-16.
[37] Al Daradkeh, D. K., et al. (2020). "The Impact of Knowledge Management Success Factors on Electronic Business in Jordanian Telecom Companies." Int. J Sup. Chain. Mgt Vol 9(6): 102-119.
[38] Alamawi, W. W., et al. (2016). "Rule Based System for Diagnosing Wireless Connection Problems Using SL5 Object." International Journal of Information Technology and Electrical Engineering 5(6): 26-33.
[39] Al-Araj, R. S. A., et al. (2020). "Classification of Animal Species Using Neural Network." International Journal of Academic Engineering Research (IJAER) 4(10): 23-31.
[40] Alfarra, A. H., et al. (2021). "An Expert System for Neck Pain Diagnosis." rnal of Academic Information Systems Research (IJAISR) 5(7): 1-8.
[41] Alfarra, A. H., et al. (2021). "Classification of Pineapple Using Deep Learning." International Journal of Academic Information Systems Research (IJAISR) 5(12): 37-41.
[42] Alghoul, A., et al. (2018). "Email Classification Using Artificial Neural Network." International Journal of Academic Engineering Research (IJAER) 2(11): 8-14.
[43] Alghoul, A., et al. (2018). "GhaydaHarb, SamyS." Abu-Naser, Email Classification Using Artificial Neural Network, International Journal of Academic Engineering Research.
[44] Al-Ghoul, M. M., et al. (2022). "Knowledge Based System for Diagnosing Custard Apple Diseases and Treatment." International Journal of Academic Engineering Research (IJAER) 6(5): 41-45.
[45] Alkahlout, M. A., et al. (2021). "Expert System Diagnosing Facial-Swelling Using CLIPS."
[46] Alkahlout, M. A., et al. (2021). "Expert System for Throat Problems Using SL5 Object." International Journal of Academic Information Systems Research (IJAISR) 5(5): 68-78.
[47] Alkahlout, M. A., et al. (2021). "Knowledge Based System for Diagnosing Throat Problem CLIPS and Delphi languages." International Journal of Academic Engineering Research (IJAER) 5(6): 7-12.
[48] Al-Kahlout, M. M., et al. (2020). "Neural Network Approach to Predict Forest Fires using Meteorological Data." International Journal of Academic Engineering Research (IJAER) 4(9): 68-72.
[49] Alkahlout, M., et al. (2021). "Classification of A few Fruits Using Deep Learning." International Journal of Academic Engineering Research (IJAER) 5(12).

[50] Jalili, R., & Shayanfar, H. A. A comparative study of sorting algorithms on different data structures. International Journal of Computer Applications, Vol. 101, No. 15, 19-25, 2019.
[51] Das, S., & Saha, A. Comparative analysis of sorting algorithms: A survey. International Journal of Computer Applications, Vol. 124 No. 5, 1-6, 2019.
[52] Yao, Z., Zhang, Y., & Zhu, Y. A comparative analysis of sorting algorithms. Journal of Physics: Conference Series, Vol. 1037, No. 1, 2022.