ALGORITHM EFFICIENCY, A SIDE-BY-SIDE COMPARISON

Radu-Mihail CIUPERCĂ[1]

Vlad-Andrei MIHAI[2]

Daniela Alexandra CRIȘAN[3]

Abstract

This study presents a comparative analysis of the efficiency of classic sorting algorithms, including Merge Sort, Quick Sort, Heap Sort, Bubble Sort, Selection Sort, Counting Sort and the enigmatic Bogo Sort. Through a series of rigorous tests on arrays of varying sizes, we measured the time complexities of each algorithm and examined their performance characteristics. Our findings reveal distinctive patterns in the behavior of these algorithms, highlighting their relative strengths and limitations in handling different data structures. By evaluating factors such as time complexity, stability, and adaptability, we provide insights that aid in the informed selection of sorting methodologies for diverse computational challenges. This study contributes to a nuanced understanding of algorithmic efficiency and provides valuable guidance for practical implementation in real-world applications.

Keywords: Efficiency, Sorting, Analysis, Complexity, Algorithms, Performance, Comparison

JEL Classification: C63

1. Introduction

Efficiency is a cornerstone in computer science, which determines how effective and efficient an algorithm is. The search for efficiency in ranking algorithms has led to the development of various methods, each with its own strengths and limitations.

This review aims to shed light on the comparison of efficiencies among classical sorting algorithms through their performance analysis under different conditions. By investigating the time complexity and practicality of algorithms, we try to find the most appropriate methods for different data structures and large arrays.

[1] Power Platform Developer at BearingPoint Romania, raduciuperca2000@gmail.com
[2] Software Developer at StarByte Romania, mihaivladandrei01@gmail.com
[3] Associate Professor, PhD, School of Computer Science for Business Management, Romanian-American University, e-mail: daniela.alexandra.crisan@rau.ro

The main contenders in this search are the time-tested algorithms Merge Sort, Quick Sort, Heap Sort, Counting Sort, Bubble Sort, Selection Sort, Counting Sort and the fun Bogo Sort. Previous research has shown that the run time complexity is strongly influenced by the programming language [1]. The present research, with rigorous testing and statistical analysis evaluation, will provide a comprehensive understanding of the misbehavior of these algorithms with respect to time complexity, stability, adapt to different data distributions.

By considering factors such as these, we try to provide valuable insights that can suggest better strategies than the best configuration methods for specific computational challenges will provide. In this comparative study, we examine the efficiency of each algorithm on arrays of different sizes, delve into the nuances of their performance that reveal the underlying complexity of their performance metrics, and target them so to provide a comprehensive review of these classic sorting algorithms, the strengths of their relatives. Let us highlight weaknesses.

2. Algorithms

2.1 Merge Sort

Merge Sort is an algorithm that is known for its efficiency and stability in sorting large lists or arrays. It is a divide-and-conquer algorithm, and it operates by recursively dividing an input array into smaller subarrays. It does this operation until each subarray consists of a single element. All of the subarrays are then merged back together in a sorted order. The main step in Merge Sort is the merging process, in which the algorithm compares the elements of the divided subarrays, and it arranges them in a sorted order.

Merge Sort has a time complexity of O(n log n) in all cases and it is very efficient, as it guarantees consistent performance. Its space complexity is O(n), ‘n’ being the number of elements in the array, but the algorithm has a stable and predictable nature that makes it a preferred choice for sorting tasks, as stability and predictability are essential for those types of tasks [2].

2.2 Quick Sort

Quick Sort is a very efficient sorting algorithm that is widely used in sorting tasks. It also follows the divide-and-conquer paradigm, and it selects a pivot element from the starting array and then it partitions the rest of the elements into two subarrays. It does this split according to whether they are less than or greater than the pivot. The subarrays are then recursively sorted, and the process is repeated until the entire array is sorted.

The Quick Sort algorithm has the ability to sort-in-place, meaning that it doesn’t require additional memory, and this is why it is very efficient. Its average time complexity is O(n log n), which makes it one of the fastest sorting algorithms. However, in the worst-case scenario, its complexity can become O(n^2), when the pivot selection is unbalanced. There are different optimization techniques like choosing the pivot strategically, contributing to mitigating the risk for improving its overall performance [3].

2.3 Heap Sort

Heap Sort is a sorting algorithm that is comparison-based. It utilizes a binary heap data structure, and it begins by creating a heap from the input array. The largest element (for a max-heap) or the smallest element (for a min-heap) is placed at the root. This element is then swapped with the last element in the heap, which is then removed from the heap and placed in the sorted array. The heap is updated, and the process repeats until we sort the entire array.

Its time complexity is O(n log n) in all cases. This makes Heap Sort an efficient and stable sorting algorithm, as it has an in-place sorting nature. It does not have the need for significant extra memory, and this is why it is a preferred choice for large data sets. However, it does not perform as well as other sorting algorithms because it has slower constant factors. Heap Sort is a valuable algorithm in situations where the data is presented as a binary heap [4].

2.4 Bubble Sort

Bubble Sort is a very simple and easy to understand algorithm. It is comparison-based and it repeatedly steps through the array, compares each pair of adjacent items and swaps them if they are in the wrong order. It repeats the passing through the array until it is sorted.

This method doesn’t make it a very efficient algorithm for large arrays. In the worst-case, its average time complexity is O(n^2), making it a slow algorithm compared to more advanced ones. For small datasets and short arrays, Bubble Sort is a practical option due to its simplicity and ease of implementation [5].

2.5 Selection Sort

Selection Sort is a comparison-based algorithm that divides the input array into a subarray of items that are already sorted and a subarray of items remaining to be sorted. It is a simple algorithm that finds the smallest (or largest, depending on the order) element from the unsorted subarray and swaps it with the leftmost unsorted element. It repeats the process until the entire array is sorted.

Like Bubble Sort, it is easy to understand and implement, but it is not considered efficient for large arrays. It has a time complexity of O(n^2) which makes it slow compared to other algorithms. However, it has the advantage of minimizing the number of swaps and it is useful in situations where the cost of swapping elements is very high [6].

2.6 Counting Sort

Counting Sort is a linear-time, non-comparative algorithm. It operates by counting the number of occurrences of each element in the input array. The input needs to consist of integers within a known range and the algorithm creates an auxiliary array that is the counting array. This counting array stores the count of each distinct element. With these counts, the algorithm determines the correct position of each element in the sorted output.

It has three phases:

- the counting phase: in which it reads the input and counts the occurrence of each element and then saves it in a counting array.
- the accumulation phase: in which it modifies the array to represent the cumulative count of elements. At this phase it ensures that each element's sorted position is accurately determined.
- sorting phase: in which it populates the output by placing each element in its correct sorted position. It does this based on the information stored in the counting array.

It has a time complexity of O(n+k), 'n' being the number of elements in the input array and 'k' being the range of possible integer values. The algorithm is efficient when the range of input values is not larger than the number of elements. Counting Sort is not suitable for sorting non-integer data or data with a wide range of values [7].

2.7 **Bogo Sort**

Bogo Sort is an algorithm that has been made for fun. It is highly inefficient and impractical, as it relies on pure luck and sheer randomness. It repeatedly shuffles the elements of the array, and it checks if they are sorted. If the array is sorted, then it stops. If it is not sorted, it does it again.

Bogo Sort is notorious for its abysmal performance. It has an average-case time complexity of O((n+1)!), but it grows factorially with the number of elements. Because of this, it is very slow and not practical for any sorting task. It is an example of how not to sort data [8].

3. Testing Methodology

To prevent bias, we chose to test all algorithms on multiple machines in a batch of one hundred tests that will be measured and stored in an external database. This test covers all 7 algorithms presented earlier and has 5 phases where an array is generated by custom method who generates random numbers based on the size of the output array, into an array that will be used il almost all algorithms. The size of an array is calculated by this formula **10^Phase Number**. The only exception is Bogo Sort, because it is an algorithm with an unclear predict rate of complexity starting with a time complexity of O (1) that can become O (∞). Based on this consideration, Bogo Sort will be run with a smaller chunk of values starting from 3 and growing up by 1 on every phase. In this scenario the formula for the size of the array is **3 + Phase Number**.

Besides this organizational part, our solution has multiple key components that facilitate the entire process from start to finish.

- C# code that executes the testing batch.
- Dataverse & Power Platform for storing and visualizing raw data.
- Power BI report to prepare data for analyzing and reporting.

3.1 C# Code

In this study we chose C# as programming language because C# is a relatively easy language to learn and use. This makes it a good choice for people who want to understand and extend the idea behind this project.

Another key point for using C# is the fact that this programming language is a strongly typed one and a compiled language too, who can help us in developing more robust code and we can also be sure that anything can be planed and tackled from the development part of the app.

In addition to these general benefits, C# also has several specific features like delegates and support with SDK for other Microsoft products like databases or web services.

3.1.1 Tracking Time

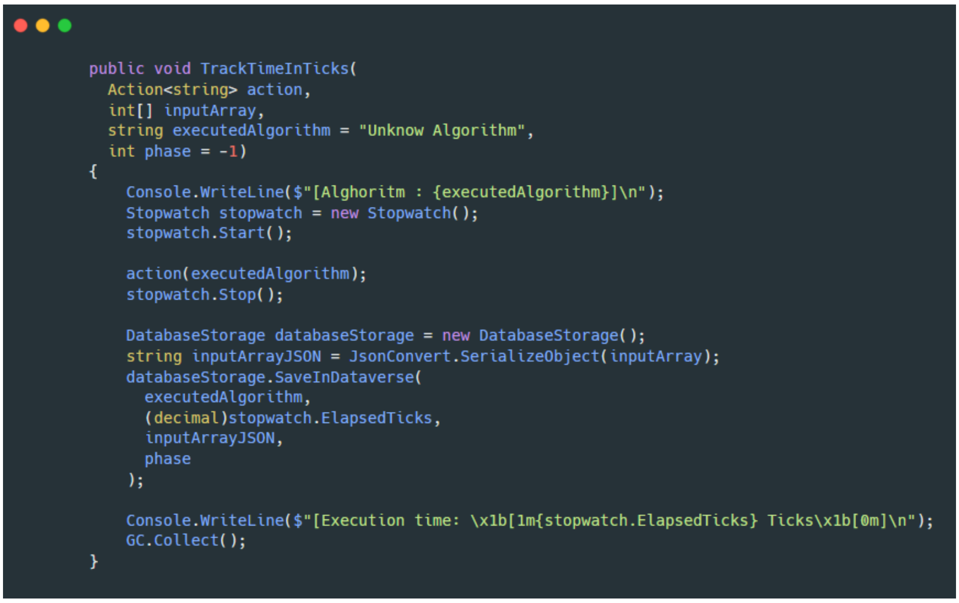
For testing we created a dedicated class where we store all the algorithm implementation, we implement a method called *TrackTimeInTicks()*. It takes four parameters:

- **action**: A delegate that points to the code that you want to measure the execution time of.
- **inputArray**: An array of integers that will be passed to the action delegate. This array is generated by a custom method created by us and will be discussed soon.
- **executedAlgorithm**: The name of the algorithm that you are measuring the execution time of. This is necessary because we need to identify the algorithms in the second phase of this article.
- **phase**: The phase of the algorithm that you are measuring the execution time of. Like execution algorithms, this information is necessary for identification of the execution phase of the algorithm.

The *TrackTimeInTicks()* method works by first creating a new Stopwatch object and starting it. Then, it calls the action delegate with the inputArray parameter that is generated using a dedicated function for that. Once the action delegate has finished executing, the Stopwatch object is stopped, and the elapsed time is calculated.

The elapsed time is then saved to a database using the DatabaseStorage class. Then, we need to serialize the inputArray parameter to JSON before saving it to the database.

Finally, the *TrackTimeInTicks()* method prints the elapsed time to the console to make it easier to watch the progress during the execution. At the end of the method, we use the built in Garbage Collection Mechanisms from C# to be sure that unused memory after execution is cleared and cannot affect other tests. In the figure below the code associated with this explanation can be seen.



```
public void TrackTimeInTicks(
  Action<string> action,
  int[] inputArray,
  string executedAlgorithm = "Unknow Algorithm",
  int phase = -1)
{
    Console.WriteLine($"[Alghoritm : {executedAlgorithm}]\n");
    Stopwatch stopwatch = new Stopwatch();
    stopwatch.Start();

    action(executedAlgorithm);
    stopwatch.Stop();

    DatabaseStorage databaseStorage = new DatabaseStorage();
    string inputArrayJSON = JsonConvert.SerializeObject(inputArray);
    databaseStorage.SaveInDataverse(
      executedAlgorithm,
      (decimal)stopwatch.ElapsedTicks,
      inputArrayJSON,
      phase
    );

    Console.WriteLine($"[Execution time: \x1b[1m{stopwatch.ElapsedTicks} Ticks\x1b[0m]\n");
    GC.Collect();
}
```

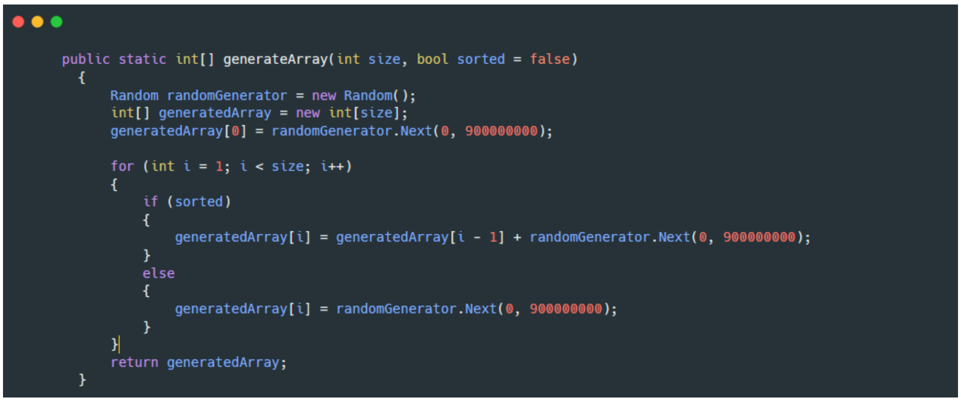
Figure 1 – Track Time in Ticks Method

3.1.2 Generating Values

The previous figure was the most important part of the project and the starting point of benchmarking a couple of classic sorting algorithms. We observed that we use another 3rd party function and classes and now we will discuss every important bit.

To generate that number, we have created a method to generate an array of numbers based on the size of the array. Numbers are generated between 0 and 900.000.000. We implemented sorted flags to generate sorted arrays for debugging purposes.

In the figure below we can see how this generation is implemented.

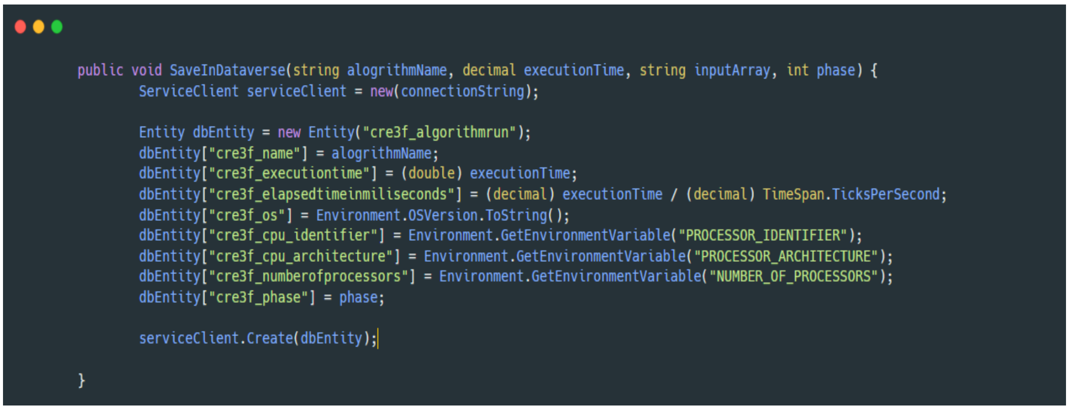


```
public static int[] generateArray(int size, bool sorted = false)
  {
      Random randomGenerator = new Random();
      int[] generatedArray = new int[size];
      generatedArray[0] = randomGenerator.Next(0, 900000000);

      for (int i = 1; i < size; i++)
      {
          if (sorted)
          {
              generatedArray[i] = generatedArray[i - 1] + randomGenerator.Next(0, 900000000);
          }
          else
          {
              generatedArray[i] = randomGenerator.Next(0, 900000000);
          }
      }
      return generatedArray;
  }
```

Figure 2 – Generate Array Method

3.1.3 Saving and storing tests

To have consistent data to be analyzed, we decided to store information into an external database because we want to run these algorithms in parallels to be efficient from time perspective and to prevent possible misleading information from reading and transferring data from console output to another data storage source. To do this we used Power Platform SDK to interact with Microsoft Dataverse as a data storage environment. Figure 3 presents how we tackled this implementation in our code.

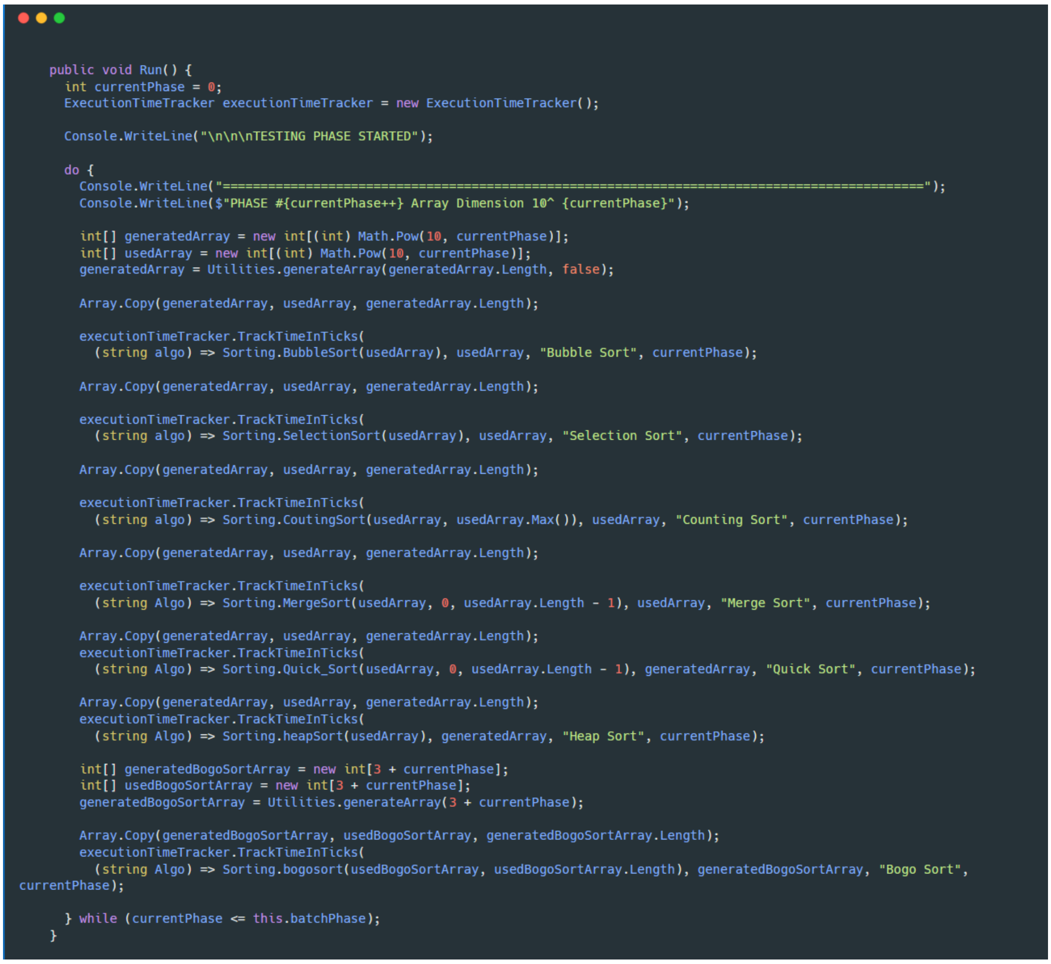


```
public void SaveInDataverse(string alogrithmName, decimal executionTime, string inputArray, int phase) {
        ServiceClient serviceClient = new(connectionString);

        Entity dbEntity = new Entity("cre3f_algorithmrun");
        dbEntity["cre3f_name"] = alogrithmName;
        dbEntity["cre3f_executiontime"] = (double) executionTime;
        dbEntity["cre3f_elapsedtimeinmiliseconds"] = (decimal) executionTime / (decimal) TimeSpan.TicksPerSecond;
        dbEntity["cre3f_os"] = Environment.OSVersion.ToString();
        dbEntity["cre3f_cpu_identifier"] = Environment.GetEnvironmentVariable("PROCESSOR_IDENTIFIER");
        dbEntity["cre3f_cpu_architecture"] = Environment.GetEnvironmentVariable("PROCESSOR_ARCHITECTURE");
        dbEntity["cre3f_numberofprocessors"] = Environment.GetEnvironmentVariable("NUMBER_OF_PROCESSORS");
        dbEntity["cre3f_phase"] = phase;

        serviceClient.Create(dbEntity);

}
```

Figure 3 – Storing data in Dataverse.

3.1.4 Test Batch and running conditions

The last piece of our puzzle is the testing batch that contains all algorithms executed through the *TrackTimeInTicks()* method. This method uses the generation function to create the initial arrays than execute each algorithm. To be sure that all algorithms are executed with the same data set we need a second array where we store the original array through execution. In Figure 4 generatedArray is the original array and usedArray is the middleman between executions. This middleman is overwriting after every execution using function method from Array class.



```
public void Run() {
  int currentPhase = 0;
  ExecutionTimeTracker executionTimeTracker = new ExecutionTimeTracker();

  Console.WriteLine("\n\n\nTESTING PHASE STARTED");

  do {
    Console.WriteLine("==============================================================================================");
    Console.WriteLine($"PHASE #{currentPhase++} Array Dimension 10^ {currentPhase}");

    int[] generatedArray = new int[(int) Math.Pow(10, currentPhase)];
    int[] usedArray = new int[(int) Math.Pow(10, currentPhase)];
    generatedArray = Utilities.generateArray(generatedArray.Length, false);

    Array.Copy(generatedArray, usedArray, generatedArray.Length);

    executionTimeTracker.TrackTimeInTicks(
      (string algo) => Sorting.BubbleSort(usedArray), usedArray, "Bubble Sort", currentPhase);

    Array.Copy(generatedArray, usedArray, generatedArray.Length);

    executionTimeTracker.TrackTimeInTicks(
      (string algo) => Sorting.SelectionSort(usedArray), usedArray, "Selection Sort", currentPhase);

    Array.Copy(generatedArray, usedArray, generatedArray.Length);

    executionTimeTracker.TrackTimeInTicks(
      (string algo) => Sorting.CoutingSort(usedArray, usedArray.Max()), usedArray, "Counting Sort", currentPhase);

    Array.Copy(generatedArray, usedArray, generatedArray.Length);

    executionTimeTracker.TrackTimeInTicks(
      (string Algo) => Sorting.MergeSort(usedArray, 0, usedArray.Length - 1), usedArray, "Merge Sort", currentPhase);

    Array.Copy(generatedArray, usedArray, generatedArray.Length);
    executionTimeTracker.TrackTimeInTicks(
      (string Algo) => Sorting.Quick_Sort(usedArray, 0, usedArray.Length - 1), generatedArray, "Quick Sort", currentPhase);

    Array.Copy(generatedArray, usedArray, generatedArray.Length);
    executionTimeTracker.TrackTimeInTicks(
      (string Algo) => Sorting.heapSort(usedArray), generatedArray, "Heap Sort", currentPhase);

    int[] generatedBogoSortArray = new int[3 + currentPhase];
    int[] usedBogoSortArray = new int[3 + currentPhase];
    generatedBogoSortArray = Utilities.generateArray(3 + currentPhase);

    Array.Copy(generatedBogoSortArray, usedBogoSortArray, generatedBogoSortArray.Length);
    executionTimeTracker.TrackTimeInTicks(
      (string Algo) => Sorting.bogosort(usedBogoSortArray, usedBogoSortArray.Length), generatedBogoSortArray, "Bogo Sort",
currentPhase);

  } while (currentPhase <= this.batchPhase);
}
```

Figure 4 – Testing batch run method

Our program executes this test batch 100 times to ensure a large quantity of data to obtain pertinent values for further analysis.

3.2 Dataverse & Power Platform Ecosystem

After we presented the code part for generating results, now it's time to show the infrastructure that facilitates storing of items and visualizations of them.

For storing data, we use Dataverse from Microsoft Power Platform. We chose this option because it is strongly linked with Power BI, another component that we chose for this project to manipulate and visualize data.

Another reason for choosing this approach is the free tier offered to developers which involves a database with a size of 2GB, always online for us. This tier is available for all organizations that have any Microsoft 365 License active.

To store data, we created a table called algorithm runs and added some key attributes to capture the relevant information of every run.

In the table below we have the name of the attribute, data type and the logical name. A logical name is the identifier that helps us in code to map the information in the storage environment.

In table 1 are presented the attributes added to our Dataverse column to support the application.

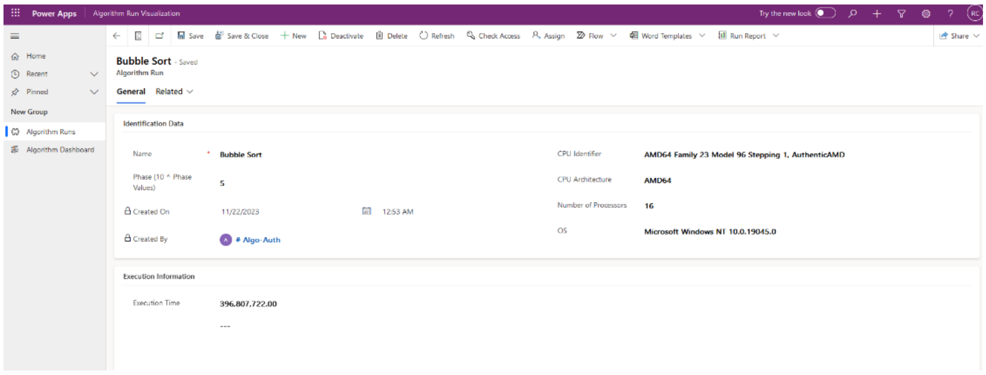
Name	**Type**	**Logical Name**
Algorithm Run	GUID - ID	cre3f_algorithmrunid
CPU Architecture	String	cre3f_cpu_architecture
CPU Identifier	String	cre3f_cpu_identifier
Elapsed Time in Seconds	Decimal	cre3f_elapsedtmeinseconds
Name (of the run algorithm)	Decimal	cre3f _name
Number of Processors	String	cre3f_numberofprocessors
OS (Operating System)	String	cre3f_OS
Phase	String	cre3f_Phase

Tabel 1 – Dataverse columns to facilitate the application

Because we want the testing batch to send data automatically, we created an Azure App Registration, and we use that as a S2S[4] User.

To monitor data submitted by the end user we also used the benefits from Power Platform free tier. We created a small **model driven app** to be able to navigate easier through the data that we collected. We have a grid for visualizing all data and a form to view and edit, if necessary, the selected data. In the figures below we have the front-end of the actual power app.

[4] S2S - Software 2 Software



Bubble Sort - Saved
Algorithm Run
General Related

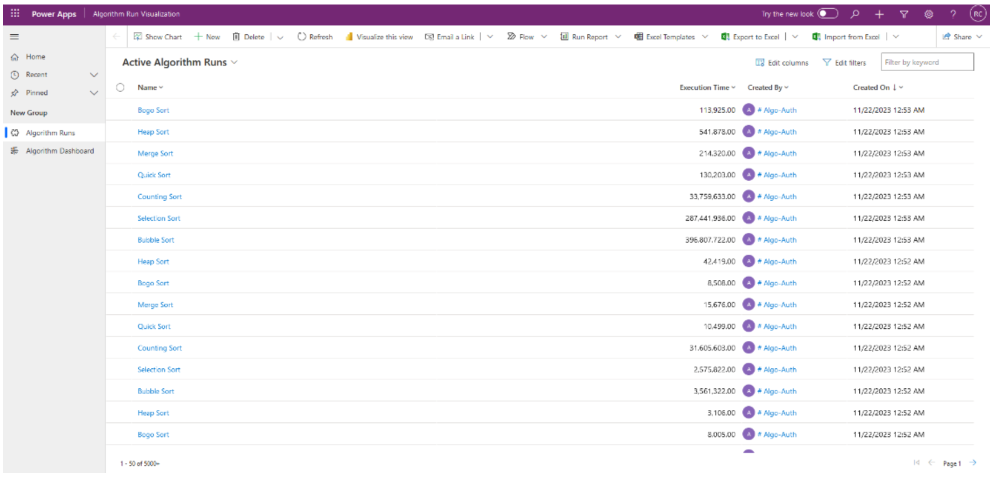
Identification Data

Name *	Bubble Sort	CPU Identifier	AMD64 Family 23 Model 96 Stepping 1, AuthenticAMD
Phase (10 ^ Phase Values)	5	CPU Architecture	AMD64
Created On	11/22/2023 12:53 AM	Number of Processors	16
Created By	# Algo-Auth	OS	Microsoft Windows NT 10.0.19045.0

Execution Information

Execution Time	396,807,722.00

Figure 5 – Power App Form for data modification



Active Algorithm Runs

Name	Execution Time	Created By	Created On ↓
Bogo Sort	113,925.00	# Algo-Auth	11/22/2023 12:53 AM
Heap Sort	541,878.00	# Algo-Auth	11/22/2023 12:53 AM
Merge Sort	214,320.00	# Algo-Auth	11/22/2023 12:53 AM
Quick Sort	130,203.00	# Algo-Auth	11/22/2023 12:53 AM
Counting Sort	33,759,633.00	# Algo-Auth	11/22/2023 12:53 AM
Selection Sort	287,441,936.00	# Algo-Auth	11/22/2023 12:53 AM
Bubble Sort	396,807,722.00	# Algo-Auth	11/22/2023 12:53 AM
Heap Sort	42,419.00	# Algo-Auth	11/22/2023 12:52 AM
Bogo Sort	8,508.00	# Algo-Auth	11/22/2023 12:52 AM
Merge Sort	15,676.00	# Algo-Auth	11/22/2023 12:52 AM
Quick Sort	10,499.00	# Algo-Auth	11/22/2023 12:52 AM
Counting Sort	31,605,603.00	# Algo-Auth	11/22/2023 12:52 AM
Selection Sort	2,575,822.00	# Algo-Auth	11/22/2023 12:52 AM
Bubble Sort	3,561,322.00	# Algo-Auth	11/22/2023 12:52 AM
Heap Sort	3,106.00	# Algo-Auth	11/22/2023 12:52 AM
Bogo Sort	8,005.00	# Algo-Auth	11/22/2023 12:52 AM

1 - 50 of 5000+ Page 1

Figure 6 – Power App Grid for data visualization

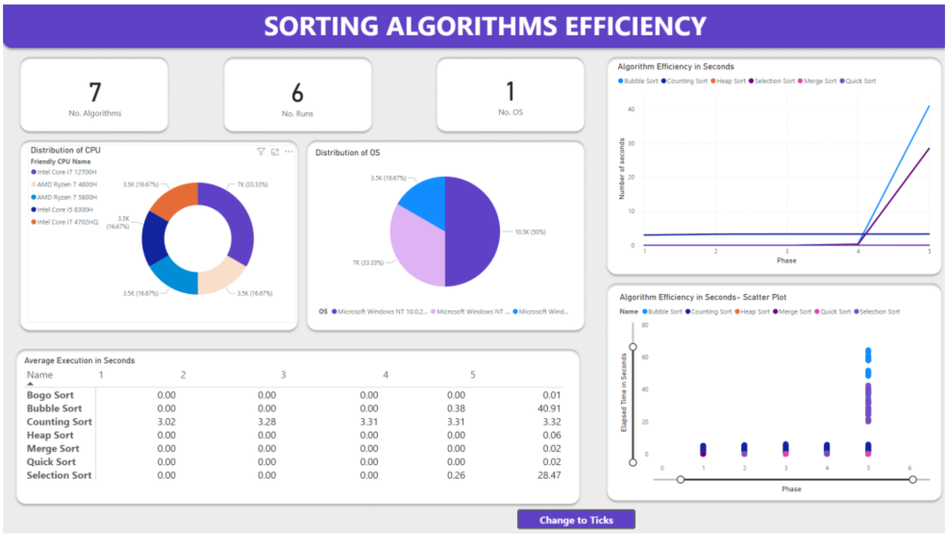
3.3 Reporting using Microsoft PowerBI

Power BI is a must-have tool for analyzing and visualizing data. Its robust and user-friendly features empower you to integrate with simple data sources, including those generated by your algorithms, and transform raw data into complex visualizations.

The platform offers several customizable visualization options like interactive dashboards, you can dynamically search for patterns, trends and outliers within your data, fostering a deeper understanding of an algorithm performance.

Because Power BI is in the same product family as Power Apps and Dataverse we have seen an opportunity to use it as an instrument for data cleaning and visualization.

In the figure below you can see how our PowerBI Report is built.


SORTING ALGORITHMS EFFICIENCY
7
No. Algorithms
6
No. Runs
1
No. OS
Algorithm Efficiency in Seconds
●Bubble Sort ●Counting Sort ●Heap Sort ●Selection Sort ●Merge Sort ●Quick Sort
Number of seconds
Phase
Distribution of CPU
Friendly CPU Name
●Intel Core i7 12700H
●AMD Ryzen 7 4800H
●AMD Ryzen 7 5800H
●Intel Core i5 8300H
●Intel Core i7 4702HQ
3.5K (16.67%)
7K (33.33%)
Distribution of OS
10.5K (50%)
OS ●Microsoft Windows NT 10.0.2... ●Microsoft Windows NT ... ●Microsoft Wind...
Algorithm Efficiency in Seconds- Scatter Plot
Name ●Bubble Sort ●Counting Sort ●Heap Sort ●Merge Sort ●Quick Sort ●Selection Sort
Elapsed Time in Seconds
Phase
Change to Ticks

Average Execution in Seconds

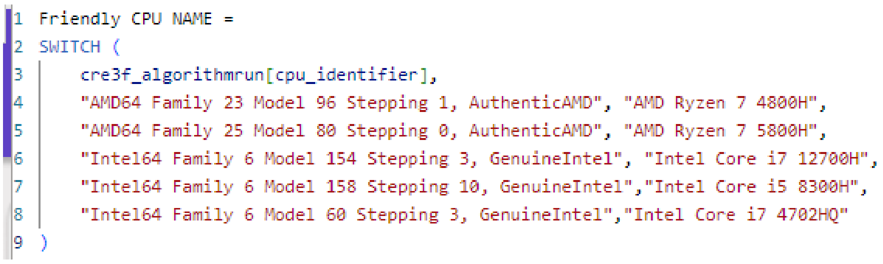
Name	1	2	3	4	5
Bogo Sort	0.00	0.00	0.00	0.00	0.01
Bubble Sort	0.00	0.00	0.00	0.38	40.91
Counting Sort	3.02	3.28	3.31	3.31	3.32
Heap Sort	0.00	0.00	0.00	0.00	0.06
Merge Sort	0.00	0.00	0.00	0.00	0.02
Quick Sort	0.00	0.00	0.00	0.00	0.02
Selection Sort	0.00	0.00	0.00	0.26	28.47

Figure 7 – Power BI report for seconds

To achieve that we need to make some transformations first. We managed to create some custom columns in PowerBI model in order to not affect the original data. To do that we have 2 alternatives **DAX** and **Power M Query** Languages. We chose to work with DAX5 because it is very similar with Microsoft Excel Formula.

One example of these modifications can be a custom column to display a more friendly name of the CPU name, because our code extract from target computer operating system from internal CPU naming offered by the manufacturer.

In figure 8 is the DAX code to do that. It's a simple SWITCH function that assigns a friendly name for the CPU identifier.



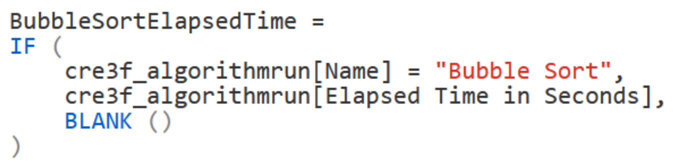
```
Friendly CPU NAME =
SWITCH (
    cre3f_algorithmrun[cpu_identifier],
    "AMD64 Family 23 Model 96 Stepping 1, AuthenticAMD", "AMD Ryzen 7 4800H",
    "AMD64 Family 25 Model 80 Stepping 0, AuthenticAMD", "AMD Ryzen 7 5800H",
    "Intel64 Family 6 Model 154 Stepping 3, GenuineIntel", "Intel Core i7 12700H",
    "Intel64 Family 6 Model 158 Stepping 10, GenuineIntel","Intel Core i5 8300H",
    "Intel64 Family 6 Model 60 Stepping 3, GenuineIntel","Intel Core i7 4702HQ"
)
```

Figure 8 – DAX Code for Friendly CPU Name column

[5] DAX - Data Analysis Expressions

Another interesting example is creating dedicated columns for creating individual controllable graph lines to interact with other elements of the report.

In figure 9 is the DAX code. In this context we look for “Bubble Sort” in the name of the run and assign elapsed time in seconds for that item, else it will set the cell as blank. We do that to prevent wrong average calculations and increase interaction capabilities.



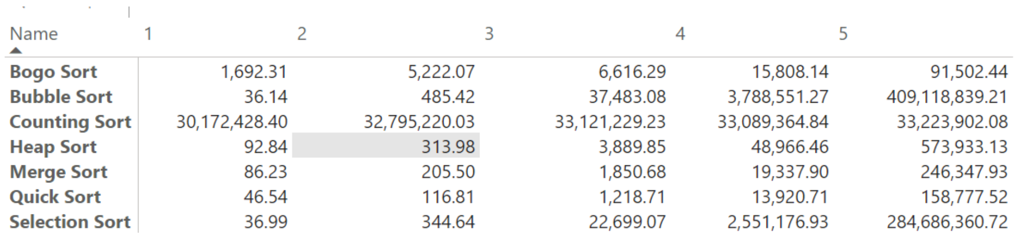
```
BubbleSortElapsedTime =
IF (
    cre3f_algorithmrun[Name] = "Bubble Sort",
    cre3f_algorithmrun[Elapsed Time in Seconds],
    BLANK ()
)
```

Figure 9 – DAX Code for Bubble Sort Elapsed Time column

4. Results

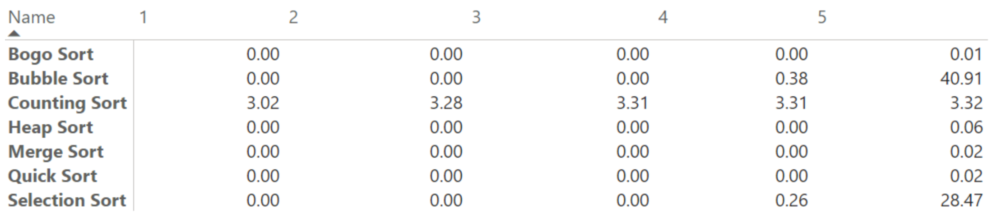
After a long journey, our results are here. Because we chose to track time in ticks in figure 10, we have all recorded values in ticks and in figure 11 the same values in seconds.

According to Microsoft Documentation, a tick is 100 nano seconds, so we obtain the value in seconds creating a custom column using DAX where we apply this formula. $Value\ in\ Seconds = Value\ in\ Ticks * 10.000.000$.



Name	1	2	3	4	5
Bogo Sort	1,692.31	5,222.07	6,616.29	15,808.14	91,502.44
Bubble Sort	36.14	485.42	37,483.08	3,788,551.27	409,118,839.21
Counting Sort	30,172,428.40	32,795,220.03	33,121,229.23	33,089,364.84	33,223,902.08
Heap Sort	92.84	313.98	3,889.85	48,966.46	573,933.13
Merge Sort	86.23	205.50	1,850.68	19,337.90	246,347.93
Quick Sort	46.54	116.81	1,218.71	13,920.71	158,777.52
Selection Sort	36.99	344.64	22,699.07	2,551,176.93	284,686,360.72

Figure 10 – Results in Ticks



Name	1	2	3	4	5
Bogo Sort	0.00	0.00	0.00	0.00	0.01
Bubble Sort	0.00	0.00	0.00	0.38	40.91
Counting Sort	3.02	3.28	3.31	3.31	3.32
Heap Sort	0.00	0.00	0.00	0.00	0.06
Merge Sort	0.00	0.00	0.00	0.00	0.02
Quick Sort	0.00	0.00	0.00	0.00	0.02
Selection Sort	0.00	0.00	0.00	0.26	28.47

Figure 11 – Results in Seconds

These values are the result of 6 runs executed in different system configurations on Windows Operating Systems for computer in a range of 8 years from release date from the manufacture and cover the most significant players in CPU markets (Intel & AMD), and covers three generations of RAM[6] memory, from DDR3 to DDR5.

One of our challenges was the similarity between Intel Core i9 12900H and Intel Core i7 12700H, because they are created on the same CPU wafer and have the same identical name. This makes our identification process almost impossible in this actual scenario, so we assimilate the Intel Core i9 12900H as Intel Core i7 12700H.

In Figure 12 we have the distribution of CPUs in our test batch.

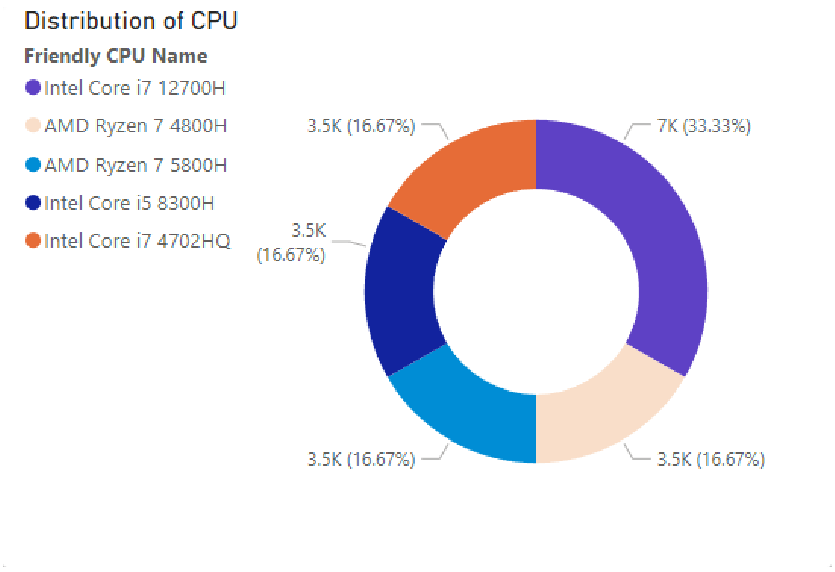


Figure 12 – Distribution of CPUs in our test batch

Also, for a better understanding of the data we attach in figure 13 and 14 there is an execution graph where are all algorithms compared. The graph has on OX axes the phase number and on OY, the number of ticks for that current execution. We can observe that in the first phase of sorting, where we have a number of 10 values, the most efficient algorithm is Bubble Sort, and the least efficient algorithm is Counting Sort. These results are in figure 13.

[6] RAM – Random access Memory

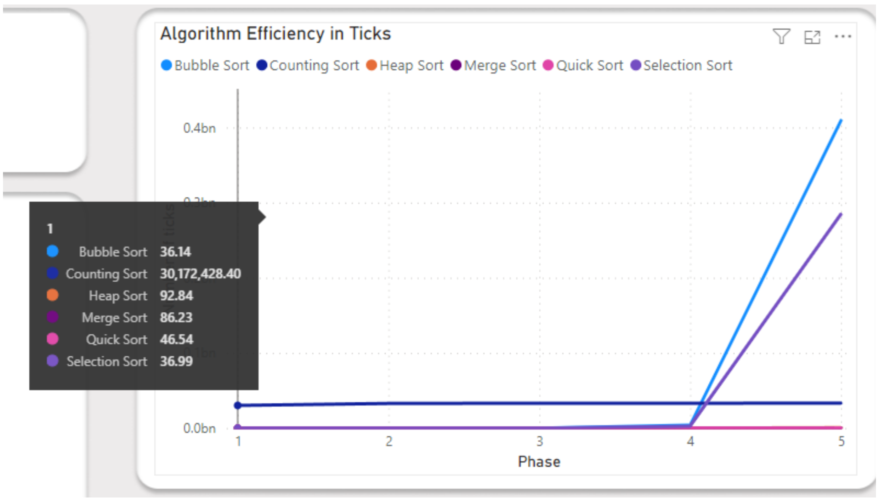


Figure 13 – Results in Seconds

In the last phase we observed a dramatic change in execution time. The fastest algorithm in phase one, now is the most inefficient, and the most effective algorithm is Quick Sort, followed by Merge Sort and Heap Sort.

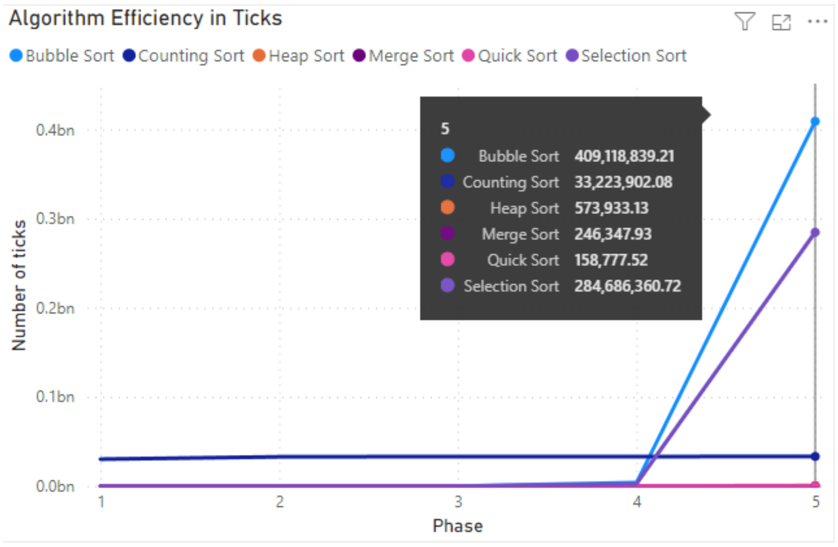


Figure 14 – Results in Seconds

If we want to compare all algorithms to the fastest one for the last phase, we observed an enormous difference between algorithms. In table 2 we have all algorithms compared one to the fastest one, Quick Sort.

Algorithm	Bubble Sort	Counting Sort	Heap Sort	Merge Sort	Quick Sort	Selection Sort
Result	257668	20924.81	361.47	155.15	100	179298.9

Tabel 2 – How slower are rest of the algorithm compared to the fastest one (in percentage)

5. Conclusions and Future Work

The presented research aimed to analyze the efficiency of some classical sorting algorithms considering a range of criteria, in order to contribute to the advancement of algorithmic understanding and to the informed selection of sequential selection methods for computational tasks.

Because our test batch was quite small, from the number of devices point of view, we can't come with pertinent conclusions for operating systems perspective and processor performance against algorithms.

We want in future to come back with an extended version of this paper in order to have more data to analyze and a more comprehensive testing and analyzing policy, with a large set of analyzed algorithms and an even distribution of operating systems. In this project we encountered lots of problems and managed to tackle and bring back some lessons learned from this process.

References

[1] Crișan D. A., Simion G. F., Moraru P. E., "Run-time analysis for sorting algorithms", Journal of Information Systems and Operations Management (JISOM), Vol 9 No 1, 2015

[2] SANCHHAYA EDUCATION PVT. LTD., "Merge Sort – Data Structure and Algorithms Tutorials," Sanchhaya Education Pvt. Ltd., [Online]. Available: https://www.geeksforgeeks.org/merge-sort/. [Accessed 15 October 2023].

[3] SANCHHAYA EDUCATION PVT. LTD., "QuickSort – Data Structure and Algorithm Tutorials," Sanchhaya Education Pvt. Ltd., [Online]. Available: https://www.geeksforgeeks.org/quick-sort/. [Accessed 18 October 2023].

[4] SANCHHAYA EDUCATION PVT. LTD., “Heap Sort – Data Structures and Algorithms Tutorials,” Sanchhaya Education Pvt. Ltd., [Online]. Available: https://www.geeksforgeeks.org/heap-sort/. [Accessed 16 October 2023].

[5] SANCHHAYA EDUCATION PVT. LTD., “Bubble Sort – Data Structure and Algorithm Tutorials,” Sanchhaya Education Pvt. Ltd., [Online]. Available: https://www.geeksforgeeks.org/bubble-sort/. [Accessed 14 October 2023].

[6] SANCHHAYA EDUCATION PVT. LTD., “Selection Sort – Data Structure and Algorithm Tutorials,” Sanchhaya Education Pvt. Ltd., [Online]. Available: https://www.geeksforgeeks.org/selection-sort/. [Accessed 15 October 2023].

[7] SANCHHAYA EDUCATION PVT. LTD., “Counting Sort – Data Structures and Algorithms Tutorials,” Sanchhaya Education Pvt. Ltd., [Online]. Available: https://www.geeksforgeeks.org/counting-sort/. [Accessed 14 October 2023].

[8] SANCHHAYA EDUCATION PVT. LTD., “BogoSort or Permutation Sort,” Sanchhaya Education Pvt. Ltd., [Online]. Available: https://www.geeksforgeeks.org/bogosort-permutation-sort/. [Accessed 23 October 2023].

Bibliography

AKINSHIN, Pro .NET Benchmarking: The Art of Performance Measurement, Apress, 2019.

CRIȘAN D. A., Simion G. F., Moraru P. E., “Run-time analysis for sorting algorithms”, Journal of Information Systems and Operations Management (JISOM), Vol 9 No 1, 2015

JAMRO M., C# Data Structures and Algorithms: Explore the possibilities of C# for developing a variety of efficient applications, Packt Publishing, 2018.

RIVERA J., Building Solutions with the Microsoft Power Platform: Solving Everyday Problems in the Enterprise, O'Reilly Media, 2023.

POWELL B., Mastering Microsoft Power BI: Expert techniques for effective data analytics and business intelligence, Packt Publishing, 2018.

MICROSOFT, “DateTime.Ticks Property | Microsoft Learn,” 14 October 2023. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/api/system.datetime.ticks?view=net-8.0.

MICROSOFT, “Ticks class | Microsoft Learn,” 14 October 2023. [Online]. Available: https://learn.microsoft.com/en-us/javascript/api/adaptive-expressions/ticks?view=botbuilder-ts-latest.

MICROSOFT Learn, “Action<T> Delegate,” Microsoft Learn, [Online]. Available: https://learn.microsoft.com/en-us/dotnet/api/system.action-1?view=net-8.0. [Accessed 10 October 2023].

MICROSOFT Learn, “Environment.GetEnvironmentVariable Method,” Microsoft Learn, [Online]. Available: https://learn.microsoft.com/en-us/dotnet/api/system.environment.getenvironmentvariable?view=net-8.0. [Accessed 21 October 2023].

SANCHHAYA EDUCATION PVT. LTD., “Counting Sort – Data Structures and Algorithms Tutorials,” Sanchhaya Education Pvt. Ltd., [Online]. Available: https://www.geeksforgeeks.org/counting-sort/. [Accessed 14 October 2023].

SANCHHAYA EDUCATION PVT. LTD., “Bubble Sort – Data Structure and Algorithm Tutorials,” Sanchhaya Education Pvt. Ltd., [Online]. Available: https://www.geeksforgeeks.org/bubble-sort/. [Accessed 14 October 2023].

SANCHHAYA EDUCATION PVT. LTD., “Selection Sort – Data Structure and Algorithm Tutorials,” Sanchhaya Education Pvt. Ltd., [Online]. Available: https://www.geeksforgeeks.org/selection-sort/. [Accessed 15 October 2023].

SANCHHAYA EDUCATION PVT. LTD., “Heap Sort – Data Structures and Algorithms Tutorials,” Sanchhaya Education Pvt. Ltd., [Online]. Available: https://www.geeksforgeeks.org/heap-sort/. [Accessed 16 October 2023].

SANCHHAYA EDUCATION PVT. LTD., “Merge Sort – Data Structure and Algorithms Tutorials,” Sanchhaya Education Pvt. Ltd., [Online]. Available: https://www.geeksforgeeks.org/merge-sort/. [Accessed 15 October 2023].

SANCHHAYA EDUCATION PVT. LTD., “QuickSort – Data Structure and Algorithm Tutorials,” Sanchhaya Education Pvt. Ltd., [Online]. Available: https://www.geeksforgeeks.org/quick-sort/. [Accessed 18 October 2023].

SANCHHAYA EDUCATION PVT. LTD., “BogoSort or Permutation Sort,” Sanchhaya Education Pvt. Ltd., [Online]. Available: https://www.geeksforgeeks.org/bogosort-permutation-sort/. [Accessed 23 October 2023].