

|  |
| --- |
|  |
| **Multi Paradigm Programming 2021** |
| By Colm Higgins ID No. G00287906 A project submitted in partial fulfilment of the requirements for the degree of: Higher Diploma in Science in Computing (Data Analytics) Galway Mayo Institute of Technology Lecturer: Dominic Carr December 2021 |
|  |
|  |
|  |

# Table of Contents

[Table of Contents ii](#_Toc89276415)

[List of Figures i](#_Toc89276416)

[List of Tables i](#_Toc89276417)

[1. Introduction 4](#_Toc89276418)

[2. Sorting Algorithms 5](#_Toc89276419)

[2.1 Simple comparison-based sort (Bubble Sort) 5](#_Toc89276420)

[2.2 Efficient comparison-based sort (Quicksort) 5](#_Toc89276421)

[3. Implementation & Benchmarking 5](#_Toc89276422)

[4. References 9](#_Toc89276423)

# List of Figures

[Figure 18: Various input sizes used for testing 5](#_Toc89276438)

[Figure 19: Graphical representation of benchmarking for sorting algorithms 6](file:///C:\Users\CMHig\OneDrive\Desktop\HDIP%20Data%20Analytics\COMPUTATIONAL%20THINKING%20WITH%20ALGORITHMS%20PROJECT%202021\COMPUTATIONAL%20THINKING%20WITH%20ALGORITHMS%20PROJECT%202021%20Colm%20Higgins.docx#_Toc89276439)

[Figure 20: Dataframe that stores average runtime for each sorting algorithm and input sizes 7](file:///C:\Users\CMHig\OneDrive\Desktop\HDIP%20Data%20Analytics\COMPUTATIONAL%20THINKING%20WITH%20ALGORITHMS%20PROJECT%202021\COMPUTATIONAL%20THINKING%20WITH%20ALGORITHMS%20PROJECT%202021%20Colm%20Higgins.docx#_Toc89276440)

[Figure 21: Choosing a sorting Algorithm 8](#_Toc89276441)

# 

# Introduction

<https://github.com/andkoc001/Multi-Paradigm-Programming/blob/main/Report%20-%20Andrzej%20Kocielski.pdf>

<https://github.com/angela1C/Multi-Paradigm-Programming-Shop-Project-2020/blob/main/report.pdf>

<https://github.com/Slawak1/MPP-Shop-Assignment/blob/main/report.pdf>

# Sorting Algorithms

## Simple comparison-based sort (Bubble Sort)

## Efficient comparison-based sort (Quicksort)

Known as one of the most important sorting algorithms to be created, Quicksort is an

# Implementation & Benchmarking

The implementation and benchmarking section of this project was undertaken using the programming language Python. Many of Pythons built in libraries and packages were utilised to carry out the testing and benchmarking of the five chosen algorithms. These include numpy (*NumPy*) that allows mathematical calculations to be performed on arrays of data. Random (*Python Random Module*) was used in the generation of the arrays of random numbers for testing of the five sorting algorithms mentioned earlier in this report. The Time module (*Python time Module (with Examples)*) was used to measure the time it took for each of the sorting algorithms to sort that random data into the correct order. Pandas (*pandas - Python Data Analysis Library*) was used to create the dataframe which comprised of the results of the benchmarking. It also allowed the results to be presented in tabular format as seen below in Figure 20. Pandas was used alongside Matplotlib.pyplot (*matplotlib.pyplot — Matplotlib 3.4.2 documentation*) to generate a graphical representation of the results which can be seen in Figure 19.

The five sorting algorithms were created as functions that required an input array as a parameter in Python. Then the main method was created to generate the input arrays for input sizes shown in Figure 18.

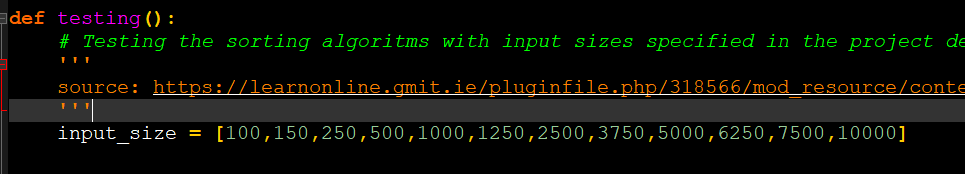


Figure 18: Various input sizes used for testing

The sorting algorithms had ten different arrays for each input size passed into them as an array that needed to be sorted. The average time taken was calculated using the Time module mentioned above and consisted of the average time taken to reorder the array over the ten instances. This was completed for each input size and provided the results shown in Figure 20 which are the results stored in the dataframe. A function was written using Matplotlib.pyplot to present the finding graphically shown in Figure 19.

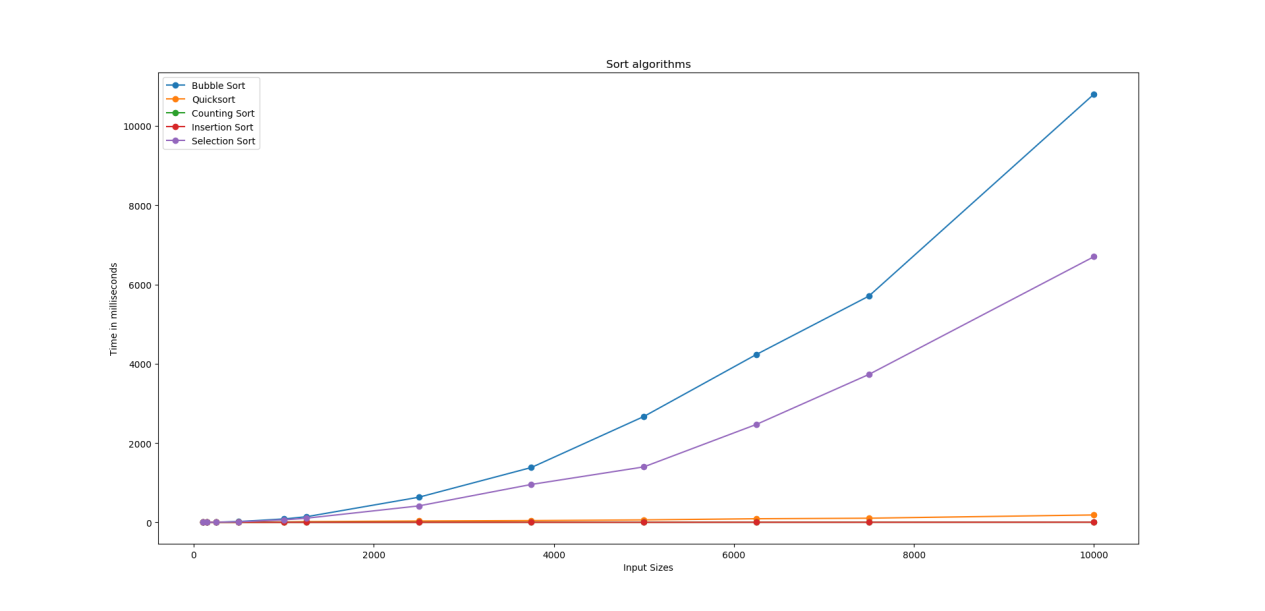
The results show the time complexity for bubble and selection sort depends heavily on the number size of the collection of data being sorted. Once the collection is greater than five-hundred to one thousand elements they struggle to compete with the other more efficient sorting algorithms. Surprisingly insertion sort performed much better than expected and the code may need to be re-examined to ensure that it is running as it is supposed to. The results for both Counting Sort and Quicksort were as expected and the performed much better with the larger collection of data.

Figure 19: Graphical representation of benchmarking for sorting algorithms

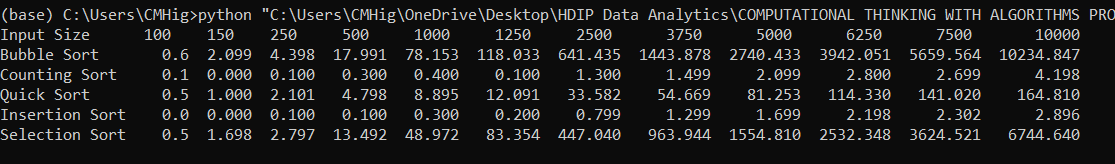


Figure 20: Dataframe that stores average runtime for each sorting algorithm and input sizes

Benchmarking for sorting algorithms is an important process that allows users to compare sorting algorithms using real data on theoretical analysis of algorithms. (Mannion, no date) It also shows that the type of sorting algorithm required for sorting data can vary depending on the needs and various inputs given.(*How to do performance micro benchmarks in Python - Peterbe.com*) Figure 21 shows some of the factors that need to be considered when choosing a sorting algorithm.

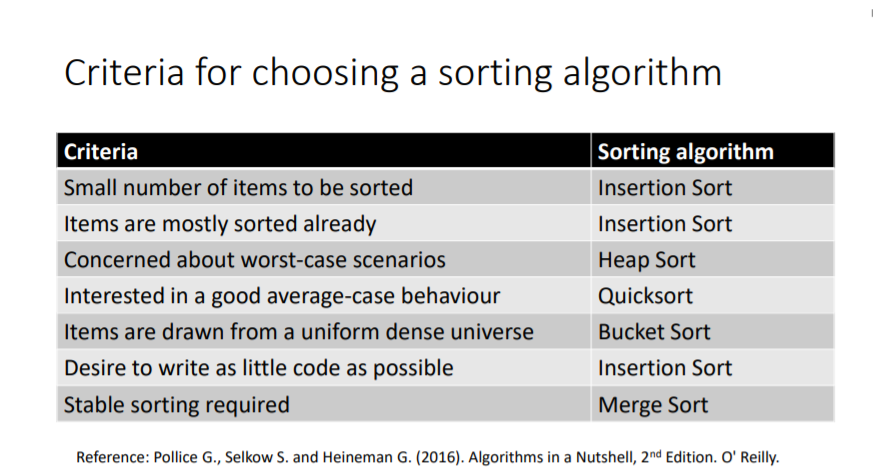


Figure 21: Choosing a sorting Algorithm

# References

*6.7. The Bubble Sort — Problem Solving with Algorithms and Data Structures* (no date).Available at: https://runestone.academy/runestone/books/published/pythonds/SortSearch/TheBubbleSort.html (Accessed: 9 May 2021).

*6.8. The Selection Sort — Problem Solving with Algorithms and Data Structures* (no date). Available at: https://runestone.academy/runestone/books/published/pythonds/SortSearch/TheSelectionSort.html (Accessed: 9 May 2021).

*6.9. The Insertion Sort — Problem Solving with Algorithms and Data Structures* (no date). Available at: https://runestone.academy/runestone/books/published/pythonds/SortSearch/TheInsertionSort.html (Accessed: 9 May 2021).

*ASCII Table: Printable Reference & Guide | OverCoded* (no date). Available at: https://www.overcoded.net/ascii-table-512119/ (Accessed: 9 May 2021).

*Counting Sort Algorithm | Studytonight* (no date). Available at: https://www.studytonight.com/data-structures/counting-sort (Accessed: 9 May 2021).

*How to do performance micro benchmarks in Python - Peterbe.com* (no date). Available at: https://www.peterbe.com/plog/how-to-do-performance-micro-benchmarks-in-python (Accessed: 12 May 2021).

Mannion, P. (no date) *Benchmarking Algorithms in Python Overview • Motivation for benchmarking • Time in Python • Benchmarking a single run • Benchmarking multiple statistical runs Motivation for benchmarking*.

*matplotlib.pyplot — Matplotlib 3.4.2 documentation* (no date). Available at: https://matplotlib.org/stable/api/\_as\_gen/matplotlib.pyplot.html (Accessed: 12 May 2021).

*NumPy* (no date). Available at: https://numpy.org/ (Accessed: 12 May 2021).

*pandas - Python Data Analysis Library* (no date). Available at: https://pandas.pydata.org/ (Accessed: 12 May 2021).

*Python Random Module* (no date). Available at: https://www.w3schools.com/python/module\_random.asp (Accessed: 12 May 2021).

*Python time Module (with Examples)* (no date). Available at: https://www.programiz.com/python-programming/time (Accessed: 12 May 2021).

*Quick Sort Algorithm | Studytonight* (no date). Available at: https://www.studytonight.com/data-structures/quick-sort (Accessed: 9 May 2021).

*Sorting Algorithms Explained with Examples in Python, Java, and C++* (no date). Available at: https://www.freecodecamp.org/news/sorting-algorithms-explained-with-examples-in-python-java-and-c/ (Accessed: 9 May 2021).

*Sorting Algorithms Part 1 COMP08033 Computational Thinking with Algorithms, Patrick Mannion GMIT* (no date).

*Sorting Techniques in Data Structures* (no date). Available at: https://www.w3schools.in/data-structures-tutorial/sorting-techniques/ (Accessed: 9 May 2021).

*string - What is lexicographical order? - Stack Overflow* (no date). Available at: https://stackoverflow.com/questions/45950646/what-is-lexicographical-order/45950665 (Accessed: 9 May 2021).

*Understanding Time and Space Complexity | by Alejandro Belgrave | Medium* (no date). Available at: https://medium.com/@whosale/understanding-time-and-space-complexity-f94b3140541b (Accessed: 9 May 2021).