Relationship between Time Complexity and Number of Elements, within different Sorting Algorithms.

Mino Karadzhov and Kestutis Dikinis

December 10th 2021
Module: SEAR
Venlo, Limburg, Netherlands

Abstract

This is the abstract. We are going to write this one at the end.

Contents

List of Figures

1 Introduction

TODO: Write about the purpose of this section, when you finish writing it.

1.1 Context and Background

In computer science, a sorting algorithm is an algorithm that takes elements of a list and puts them into an order - ascending or descending. The most popular values used are numerical and lexicographical order. It is important to have efficient sorting for optimising other algorithms, such as merge and search algorithms, as they require data to be in sorted list. Whenever there is a problem to be solved, there is many ways to approach it. Even though all those approaches end up with the same result, the time it takes for each approach can be vastly different. And the best example of that is sorting algorithms. - History of Algorithms.

1.2 Modern Usage, developments

Sorting algorithms are everywhere in day-to-day life in digital world. Everywhere starting from file browser in a computer - you can sort files on any parameters. Galleries in mobile phones - images sorted by date, contacts sorted alphabetically . Searching for a product in online store will have an option to sort products in many ways. Commercial computing: government organisations, financial institutions, and commercial enterprises organise much of this information by sorting it. Whether the information is accounts to be sorted by name or number, transactions to be sorted by time or place, mail to be sorted by postal code or address, files to be sorted by name or date, or whatever, processing such data is sure to involve a sorting algorithm somewhere along the way. Numerical computations: Scientific computing is often concerned with accuracy (how close are we to the true answer?). Accuracy is extremely important when we are performing millions of computations with estimated values such as the floating-point representation of real numbers that we commonly use on computers. Some numerical algorithms use priority queues and sorting to control accuracy in calculations.All in all, all any and every kind of list there is has been sorted in some way or another, as it is hard for a human to make use of unsorted data.[2]

1.3 Sorting algorithm explanations

Bubblesort is a simple sorting algorithm that belongs to the family of comparison sorting. It works by repeatedly stepping through the list to be sorted, comparing two items at a time and swapping them if they are in the wrong order. Bubblesort has a worst-case complexity O(n 2) and in the best case O(n). Its memory complexity is O(1). [2]
Heapsort is a comparison-based sorting algorithm, and is part of the Selection sort family. Although somewhat slower in practice on most machines than a good implementation of Quicksort, it has the advantage of a worst-case O(n log n) runtime.[2]
Mergesort belongs to the family of comparison-based sorting. It has an average and worst-case performance of O(n log n). Unfortunately, Mergesort requires three times the memory of in-place algorithms such as Insertionsort. [2]

Selection sort belongs to the family of in-place comparison sorting. It typically searches for the minimum value, exchanges it with the value in the first position and repeats the first two steps for the remaining list. On average Selection sort has a O(n 2) complexity that makes it inefficient on large lists. Selection sort typically outperforms Bubblesort but is generally outperformed by Insertionsort.[2]
On the other side of the sorting algorithm spectrum there is BogoSort. It was designed as a joke about how not to design sorting algorithms. BogoSort takes and randomly swaps places all elements of the list and repeats it until the list is sorted. Complexity of BogoSort algorithm is O((n+1)!).

1.4 Hypothesis, Problems and Considerations

- Our Hypothesis that we are going to see a linear growth of the Time Complexity, towards Size of Collection. - What is Time Complexity ? - Why is Time Complexity important? - What could affect the Time Complexity. - External factors that can affect the Time Complexity.

1.4.1 Time Complexity and the importance of it.

1.4.2 Factors, operating on Time Complexity.

1.4.3 Different methods of approaching our Research question.

2 Methodology

In order to approve or deny our initial hypothesis, an Empirical Analysis of certain Sorting Algorithms was conducted. This chapter of the paper describes the process of conducting the experiment and creating the proper experimental environment. The subject group of Sorting Algorithms includes: QuickSort, BubbleSort, SelectionSort and HeapSort. The main goal of the analysis is to observe changes of the Time Complexity, when the same algorithm is given the task to sort a collection of a certain size. Each observation of our experiment includes a task of sorting, performed by one of the sorting algorithms from our subject group, performed on a collection of certain size.
In order to conduct the Empirical Analysis, a custom-made Java application was developed and used in the process. The main purpose of which is to create a proper experimental environment , that is going to allow close observation of each algorithm from our subject group.
The overall goal is to gain an output, which allows data analysis on the covariation between the Size of Collection, used for the observation, and the time that was required for the algorithm to fully perform the sorting task.

2.1 Working Implementation of our Algorithms.

One requirement of the Empirical Analysis is a working implementation of the Sorting Algorithms, that are going to be used for the experiment. Due to this, the Java programming language was used to create working implementations of each one of the four members of our

Subject group. All of the concrete implementations of the Algorithms implement a common interface, towards which the testing is being conducted.

2.2 Developing and using our experimental environment.

In order to serve the need of a proper experimental environment for conducting the Analysis, Petko was developed. Petko is the name of the custom-made Algorithm Analysis tool, that generates unsorted collections of a fixed size and uses sorters(Sorting Algorithms) to conduct the sorting operation. The latter is being closely monitored and the time required for each sorting is being recorded in an output file, in a format of nanoseconds.
Each Algorithm is used to perform 18 different observations. Each Observation requires the algorithm to sort an unsorted collection of a fixed size. The size of the unordered collection grows with each following observation. The first observation of each algorithm starts with performing a sorting on a Collection with a number of elements of 2, whereas the last(18th) observation provides the Algorithm with an unsorted collection of 262144 elements. The growth of Collection Size is Logarithmic with base 2. Respectively, this means that the first collection for sorting is with size 2^1, whereas the collection for the last observation has a size equal to 2^{18}.
Petko actively uses the Java.time integrated library, in order to measure the Duration of time between the start and end of the sorting process. That is made possible, by placing an Instant at the Start and End point of each Algorithm execution. The following PseudoCode provides a brief description of the technique:

```
$startPoint \leftarrow Instant.now()$
$sort()$
$endPoint \leftarrow Instant.now()$
$timeComplexity \leftarrow Duration.between(startPoint, endPoint)$
```

Each Observation records the Sorting Algorithm that is being used, the TimeComplexity of the observation and the Size of Collection. This output is being gathered and recored into an external File under the .csv extension.

2.3 Data Analysis and Visualisation

Petko's output file provides a great base for conducting Data Analysis on the results of the observations. By doing this, it is possible to identify and study certain patterns in the covariation between Time Complexity and Size of Collection. The same output file was imported into R and the findings were visualised, thanks to the tidyverse library.

3 Results

This section provides a detailed overview of the experimental results.

3.1 Experimental Results

After conducting the Emperical Analysis, 72 datapoints were generated. The same visualised on a ScatterPlot have the following view:

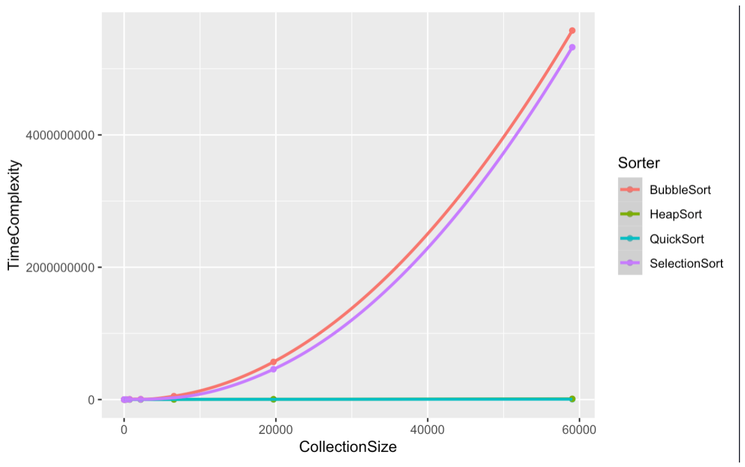

Figure 1: Overall view of the Experimental results.

As we can clearly see from the above diagram, the relationship between Time Complexity and Collection Size differes between each one of the Sorters. In addition to this, we can easily identify a sharp growth of BubbleSort and SelectionSort time complexities, when the Collection size becomes more than 20 000 elements. On the other hand, looking at the QuickSort and HeapSort, we can easily observe a relationship, which can be described as similliar to Constant Time Complexity. Taking the above into account, we can easily identify 2 sub-groups of sorters - One, where TimeComplexity appears to be constant and One, where we can clearly see a Time Complexity growth, similliar to a Linear Growth.
Since the two sub-groups are vastly contrasting on the Overall view, this may be a reason why we are missing certain patterns on our visualisation.

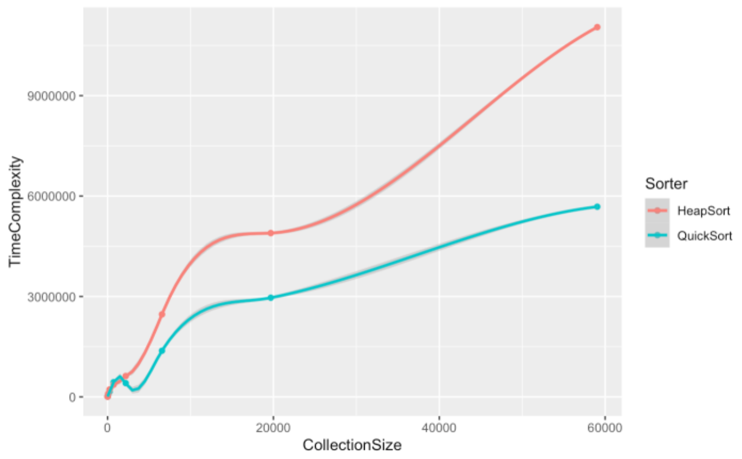

Figure 2: Focused view on the sub-group with lower time complexity.

After creating a separate visualisation for the sub-group with lower Time Complexity, we can see more regarding the growth of Time Complexity, as the overlapping on the overal view did not provide a high level of detail. From the focuesed view, we clearly see that the Sorter with the lowest Time Complexity is the QuickSort, which is being followed by the other member of the sub-group - HeapSort. In addition to this, we can spot a rapid growth in the TImeComplexity, where the CollectionSize is between 0 and 10 000. The same growth appears to be substantially bigger in the observations, that include HeapSort.

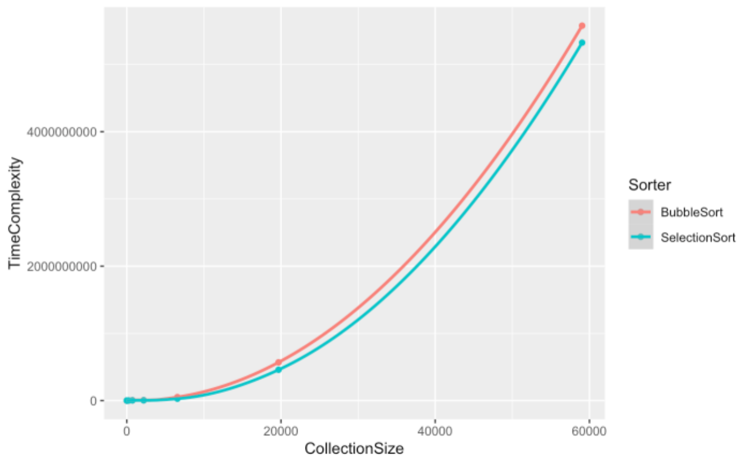

Figure 3: Focused view on the sub-group with higher time complexity

Taking a focused look at the sub-group with higher time complexity, we can see that both of the sorters have a similar growth of Time Complexity, that leaves BubbleSort with slightly higher value of Complexity. The Growth here is similliar to a linear one and closer to the one that can be viewed on the Overall results view.

4 Discussion

- Small intro on the purpouse of the section here.

4.1 Interpretation

4.2 Considerations

This subsection describes the Considerations made during the Experiment, as well as reasoning for some of the decisions described in the Methodology section.

4.2.1 Limitations of the Empirical type of Analysis

It has to be taken into account that the Empirical way of conducting the Analysis is not the best, in terms of time efficiency. That is due to the fact that this type of analysis, requires a working implementation of the subjects(Sorting Algorithms) that are going to be analysed. Furthermore, a certain amount of computing power is required, as this type of analysis requires the Algorithms to be put into action. In order to experiment with bigger Collections, which could be beneficial for revealing (otherwise hidden) patterns, larger amount of computing power is going to be required.

4.2.2 Limitations of the Collection Size

Because of the nature of Empirical Analysis, described in the last sub section, this experiment used Collections of size, just up to 2^{18} - 262144 elements.This limitation was present, mainly due to the fact that the machine, we used for testing, was not capable of experimenting with bigger collections. Would we have more computing power at our disposal, there could've been more patterns and datapoints to extend the scope of the experiment.

4.2.3 Limitations of the Machine, used for the Experiment

The described experiment was conducted on a Machine with the following technical specifications : CPU: 2 GHz Quad-Core Intel Core i5, 16 GB 3733 MHz LPDDR4X RAM. In addition to the Collection Size limitation, it is worth considering the fact that our experimental results were somehow influeced by the computational power, available with the above specifications. This means, that there is a high chance of observing different time complexity values, if the same experiment is conducted on another machine.

4.3 Evaluations

As this experiment is being conducted in a relatively small scope, taking into account the limited size of collections used, there could be an argument that this limits us from seeing how the Relationship between Time Complexity and Size of Collection changes with having bigger collections for sorting.