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. - History of Algorithms.
- Modern Usage, developments. - Maybe some discussion on sorting algorithms(if present somewhere and can be quoted).

1.2 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.

1.2.1 Time Complexity and the importance of it.

1.2.2 Factors, operating on Time Complexity.

1.2.3 Different methods of approaching our Research question.

2 Methodology

In order to approve or deny our initial hypothesis, an Emperical Analysis of certain Sorting Algorithms was conducted. This chapter of the paper describes the process of conducting the experiment and creating the proper experimental enviornment. 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 Emperical Analysis, a custom-made Java application was developed and used in the process. The main purpouse of which is to create a proper experimental enviornment, that is going to allow close observation of each algorithm from our sunject 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 perfrom the sorting task.

2.1 Working Implemntation 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 enviornment 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.
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 libray, 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

- Small intro on the section here

3.1 Experimental Results

4 Discussion

- Small intro on the purpouse of the section here.

4.1 Interpretation

- Extended analysis on our graphs - Deny or Approve our Hypothesis(Partially aprove it)

4.2 Considerations

- Limitations of the Machine used - Limitations of the Emperical type of Analysis

4.3 Evaluations

- Simple Evaluation of our findings