COMPARISON STUDY OF SORTING TECHNIQUES IN STATIC DATA STRUCTURE

ANWAR NASER FRAK

UNIVERSITI TUN HUSSEIN ONN MALAYSIA
COMPARISON STUDY ON SORTING TECHNIQUES IN STATIC DATA STRUCTURE

ANWAR NASER FRAK

A dissertation submitted in partial fulfilment of the requirement for the award of the Degree of Master of Computer Science (Software Engineering)

Faculty of Computer Science and Information Technology
Universiti Tun Hussein Onn Malaysia

MARCH 2016
To my beloved father and mother

This dissertation is dedicated to my father, who taught me that the best kind of knowledge to have is that which is learned for its own sake. It is also dedicated to my beloved mother, who taught me that even the largest task can be accomplished if it is done one step at a time.

Thank you for your love and support
ACKNOWLEDGMENT

First and foremost, I would like to thank the almighty God (ALLAH S.W.T) for all the abilities and opportunities He provided me through all my life and His blessings that enables me to do this research.

It is the greatest honor to be able to work under supervision of Dr. Mohd Zainuri Bin Saringat. I am grateful to be one of the luckiest persons who had a chance to work with him. I am thankful and gratified for all of his help, assistance, inspiration and guidance on the all aspects beside his patience and understanding.

I wish to express my sincere gratitude to everyone who contributed to the successful completion of my study. I would like to express my gratitude to Universiti Tun Hussein Onn Malaysia (UTHM) for all support through my master.
ABSTRACT

To manage and organize large data is imperative in order to formulate the data analysis and data processing efficiency. Thus, to handle large data becomes highly enviable, whilst, it is premised that the sorting techniques eliminate ambiguities with less effort. Therefore, this study investigates the functionality of a set of sorting techniques to observe which technique to provide better efficiency in terms of sorting data. Therefore, five types of sorting techniques of static data structure, namely: Bubble, Insertion, Selection in group $O(n^2)$ complexity and Merge, Quick in group $O(n \log n)$ complexity using the C++ programming language have been used. Each sorting technique was tested on four groups between 100 and 30000 of dataset. To validate the performance of sorting techniques, three performance metrics which are time complexity, execution time (run time) and size of dataset were used. All experimental setups were accomplished using simple linear regression where experimental results illustrate that Quick sort is more efficiency than Merge Insertion, Selection and Bubble sort based on run time and size of data using array and Selection sort is more efficient than Bubble and Insertion in large data size using array. In addition, Bubble, Insertion and Selection have good performance for small data size using array while Merge and Quick sort have good performance in large data size using array and sorting technique with good behavior $O(n \log n)$ more efficient rather than sorting technique with bad behavior is $O(n^2)$ using array.
Mengurus dan mengatur kuantiti data yang banyak adalah penting untuk merumuskan analisis data dan kecekapan proses data. Dengan itu, untuk mengendalikan kuantiti data yang banyak, adalah lebih baik sekiranya teknik susunan dapat menghapuskan kesamaran dengan usaha yang minimum. Oleh itu, kajian ini menyiasat satu fungsi set teknik susunan untuk memerhatikan teknik mana yang dapat menyediakan kecekapan yang lebih baik. Kerana itu, lima jenis teknik susun struktur data statik, iaitu: Bubble, Insertion, Selection dalam kumpulan O (n^2) manakala Merge dan Quick dalam kumpulan O (n log n) yang menggunakan bahasa C++ telah pun digunakan. Setiap teknik susunan telah diuji dalam empat kumpulan di antara 100 dan 30000 data. Untuk mengesahkan prestasi dari teknik penyusun, tiga metrik telah digunakan iaitu kerumitan dalam masa, pelaksanaan masa dan saiz data. Semua persediaan eksperimen telah pun selesai menggunakan regresi linear sederhana. Keputusan eksperimen menunjukkan bahawa Quick dan Merge adalah lebih cekap daripada Insertion, Selection dan Bubble adalah lebih cekap berdasarkan masa dan saiz data menggunakan tata susunan dan Selection adalah lebih cekap daripada Bubble dan Insertion pada saiz data besar menggunakan tata susunan. Di samping itu, Bubble, Insertion dan Selection mempunyai prestasi yang baik untuk saiz data kecil menggunakan tata susunan manakala Merge dan Quick mempunyai prestasi yang baik dalam saiz data yang besar menggunakan tata susunan dan teknik susunan dengan tingkah laku O (n log n) yang lebih cekap daripada teknik susunan dengan tingkah laku adalah O (n^2) menggunakan tata susunan.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>TITLE</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>DECLARATION</td>
<td>ii</td>
</tr>
<tr>
<td>DEDICATION</td>
<td>iii</td>
</tr>
<tr>
<td>ACKNOWLEDGMENT</td>
<td>iv</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>v</td>
</tr>
<tr>
<td>ABSTRAK</td>
<td>vi</td>
</tr>
<tr>
<td>TABLE OF CONTENTS</td>
<td>vii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>xii</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>xiv</td>
</tr>
<tr>
<td>LIST OF APPENDICES</td>
<td>xvii</td>
</tr>
</tbody>
</table>

## CHAPTER 1 INTRODUCTION

1.1 Background of Study  1
1.2 Motivation  2
1.3 Research Objectives  3
1.4 Research Scope  3
1.5 Thesis Outline  3

## CHAPTER 2 LITERATURE REVIEW

2.1 Overview  4
2.2 Data Structures  4
2.3 Static Array data structures  5
   2.3.1 Advantages of using Array  6
   2.3.2 Disadvantages of using Array  6
2.4 Execution time  6
2.5 Running Time Analysis  7
2.6 Time complexity  7
   2.6.1 Worst-Case analysis  8
   2.6.2 Best-Case analysis  8
### 2.6.3 Average Case analysis  

### 2.7 Big-O Notation  

### 2.8 Sorting  
- 2.8.1 Quicksort  
- 2.8.2 Selection sort  
- 2.8.3 Insertion sort  
- 2.8.4 Bubble sort  
- 2.8.5 Merge sort  

### 2.9 Simple Linear Regression  

### 2.10 Related work  

### 2.11 Summary  

#### CHAPTER 3 RESEARCH METHODOLOGY  

- 3.1 Overview  
- 3.2 Proposed Framework  
- 3.3 Phase 1: Implementing the Sorting Techniques  
- 3.4 Phase 2: Calculating the Complexity of Sorting Techniques  
- 3.5 Phase 3: Comparative Analysis  
- 3.6 Summary  

#### CHAPTER 4 IMPLEMENTATION AND ANALYSIS  

- 4.1 Overview  
- 4.2 Implementations of Sorting Technique  
  - 4.2.1 Description of Test Data Sets  
  - 4.2.2 Create and Open File  
  - 4.2.3 Insert data file into array  
  - 4.2.4 The implementatio Menu  
- 4.3 Calculating the Complexity of Sorting Algorithms  
  - 4.3.1 Complexity of bubble sort technique  
  - 4.3.2 Complexity of insertion sort technique  
  - 4.3.3 Complexity of selection sort technique  
  - 4.3.4 Complexity of merge sort technique  
  - 4.3.1.5 Complexity of quick sort technique
4.4 The Results of Efficiency Measurements Four Group Data

4.4.1 The Execution Time Results of Bubble Sort Technique

4.4.1.1 The Execution Time Results of Bubble Sort Group 1 (100 - 1,000) 34

4.4.1.2 The Execution Time Results of Bubble Sort Group 2 (2,000 - 10,000) 36

4.4.1.3 The Execution Time Results of Bubble Sort Group 3 (11,000 - 20,000) 38

4.4.1.4 The Execution Time Results of Bubble Sort Group 4 (21,000 - 30,000) 40

4.4.2 The Execution Time Results of Insertion Sort Technique

4.4.2.1 The Execution Time Results of Insertion Sort Group 1 (100 - 1,000) 42

4.4.2.2 The Execution Time Results of Insertion Sort Group 2 (2,000 - 10,000) 44

4.4.2.3 The Execution Time Results of Insertion Sort Group 3 (11,000 - 20,000) 46

4.4.2.4 The Execution Time Results of Insertion Sort Group 4 (21,000 - 30,000) 48

4.4.3 The Execution Time Results of Selection Sort Technique

4.4.3.1 The Execution Time Results of Selection Sort Group 1 (100 - 1,000) 50

4.4.3.2 The Execution Time Results of Selection Sort Group 2 (2,000 - 10,000) 52

4.4.3.3 The Execution Time Results of Selection Sort Group 3 (11,000 - 20,000) 54

4.4.3.4 The Execution Time Results for Selection Sort Group 4 (21,000 - 30,000) 56

4.4.4 The Execution Time Results of Merge Sort Technique 58
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4.4.1 The Result of Execution Time for Merge Sort Group 1 (100-1,000)</td>
<td>58</td>
</tr>
<tr>
<td>4.4.4.2 The Execution Time Results for Merge Sort Group 2 (2,000-10,000)</td>
<td>60</td>
</tr>
<tr>
<td>4.4.4.3 The Execution Time Results of Merge Sort Group 3 (11,000-20,000)</td>
<td>62</td>
</tr>
<tr>
<td>4.4.4.4 The Execution Time of Merge Sort Group 4 (21,000-30,000)</td>
<td>64</td>
</tr>
<tr>
<td>4.4.5 The Execution Time Results of Quick Sort Technique</td>
<td>66</td>
</tr>
<tr>
<td>4.4.5.1 The Execution Time Results of Quick Sort Group 1 (100-1,000)</td>
<td>66</td>
</tr>
<tr>
<td>4.4.5.2 The Execution Time Results of Quick Sort Group 2 (2000-10000)</td>
<td>68</td>
</tr>
<tr>
<td>4.4.5.3 The Execution Time Results of Quick Sort Group 3 (11,000-20,000)</td>
<td>70</td>
</tr>
<tr>
<td>4.4.5.4 The Execution Time Results of Quick Sort Group 4 (21000-30000)</td>
<td>72</td>
</tr>
<tr>
<td>4.5 Comparative Analysis</td>
<td>74</td>
</tr>
<tr>
<td>4.6 Summary</td>
<td>82</td>
</tr>
</tbody>
</table>

**CHAPTER 5 CONCLUSION**

5.1 Introduction 83

5.2 Achievements of Objectives 83

5.2.1 Implementation of Five Sorting Techniques 84

5.2.2 Calculation of Complexity 84

5.2.3 Comparative Analysis of Results Based on Efficiency 84

5.3 Contribution 85

5.4 Summary and Future Work 86

**REFERENCES** 87

**APPENDIX** 90
LIST OF FIGURES

2.1 Big O Notation Graph 9
3.1 The Four Phases of the Study 18
3.2 Research Framework 19
4.1 The Menu of Implementation 26
4.2 Experimental Result of Bubble Sort for Group1 34
4.3 Experimental Result of Bubble Sort for Group2 36
4.4 Experimental Result of Bubble Sort for Group3 38
4.5 Experimental Result of Bubble Sort for Group4 40
4.6 Experimental Result of Insertion Sort for Group1 42
4.7 The Experimental Result of Insertion Sort for Group2 44
4.8 the experimental results of insertion sort of group 3 46
4.9 The Experimental Result of Insertion Sort of Group 4 48
4.10 The Experimental Result of Selection Sort of Group1 50
4.11 The Experimental Result Of Selection Sort for Group2 52
4.12 The Experimental Rresult of Selection Sort of Group 3 54
4.13 The Experimental Result of Selection Sort for Group4 56
4.14 The Experimental Rresult of Merge Sort of Group 1 58
4.15 The Experimental Results of Merge Sort of Group 2 60
4.16 The Experimental Result of Merge Sort of Group 3 62
4.17 The Experimental Result of Merge Sort of Group 4 64
4.18 The Experimental Result of Quick Sort of Group 1 66
4.19 Extract classes and interfaces information 68
4.20 The Experimental Result of Quick Sort of Group 3
70
4.21 The Experimental result of Quick sort of Group 4
72
4.22 The Compared of estimated value of Sorting Group1
75
4.23 The Comparison of estimated value of sorting Group2
76
4.24 The Compared of estimated value of sorting Group3
77
4.25 The Comparison of estimated value of sorting Group 4
77
4.26 The Comparison of average estimated value of five sorting
Techniques
78
4.27 Comparison of the average of ratio between sorting
between sorting
80
4.28 Comparison of the average of speed ratio
81
LIST OF TABLES

2.1 Time Complexity Algorithm 8
2.2 work related 15
4.1 Data set groups 23
4.2 Create and open file complexity 24
4.3 Complexity Insert file data to array 25
4.4 Bubble Sort Complexity 28
4.5 Insertion Sort Complexity 29
4.6 Selection Sort Complexity 30
4.7 Merge Sort Complexity 31
4.8 Quick Sort Complexity 32
4.9 The Experimental Result Of Execution Time of Bubble Sort For Data Set Group 1 (100-1000) 35
4.10 The Experimental Result Of Execution Time of Bubble Sort For Data Set Group2 (2000-10000) 37
4.11 The Experimental Result Of Execution Time of Bubble Sort For Data Set Group 3 (11000-20000) 39
4.12 The Experimental Result Of Execution Time of Bubble Sort For Data Set Group 4 (21000-30000) 41
4.13 The Experimental Result Of Execution Time of Insertion Sort For Data Set Group 1 (100-1000) 43
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.14</td>
<td>The Experimental Result Of Execution Time of Insertion Sort For Data Set Group 2 (2000-10000)</td>
</tr>
<tr>
<td>4.15</td>
<td>The Experimental Result Of Execution Time of Insertion Sort For Data Set Group 3 (11000-20000)</td>
</tr>
<tr>
<td>4.16</td>
<td>The Experimental Result Of Execution Time of Insertion Sort For Data Set Group 4 (21000-30000)</td>
</tr>
<tr>
<td>4.17</td>
<td>The Experimental Result Of Execution Time of Selection Sort For Data Set Group 1 (100-1000)</td>
</tr>
<tr>
<td>4.18</td>
<td>The Experimental Result Of Execution Time of Selection Sort For Data Set Group 2 (2000-10000)</td>
</tr>
<tr>
<td>4.19</td>
<td>The Experimental Result Of Execution Time of Selection Sort For Data Set Group 3 (11000-20000)</td>
</tr>
<tr>
<td>4.20</td>
<td>The Experimental Result Of Execution Time of Selection Sort For Data Set Group 4 (21000-30000)</td>
</tr>
<tr>
<td>4.21</td>
<td>The Experimental Result Of Execution Time of Merge Sort For Data Set Group 1 (100-1000)</td>
</tr>
<tr>
<td>4.22</td>
<td>The Experimental Result Of Execution Time of Merge Sort For Data Set Group 2 (2000-10000)</td>
</tr>
<tr>
<td>4.23</td>
<td>The Experimental Result Of Execution Time of Merge Sort For Data Set Group 3 (11000-20000)</td>
</tr>
<tr>
<td>4.24</td>
<td>The Experimental Result Of Execution Time of Merge Sort For Data Set Group 4 (21000-30000)</td>
</tr>
<tr>
<td>4.25</td>
<td>The Experimental Result Of Execution Time of Quick Sort For Data Set Group 1 (100-1000)</td>
</tr>
<tr>
<td>4.26</td>
<td>The Experimental Result Of Execution Time of Quick Sort For Data Set Group 2 (2000-10000)</td>
</tr>
<tr>
<td>4.27</td>
<td>The Experimental Result Of Execution Time of Quick Sort For Data Set Group 3 (11000-20000)</td>
</tr>
</tbody>
</table>
4.28 The Experimental Result Of Execution Time of Quick Sort For Data Set Group4(21000-30000) 73
4.29 The Comparative Analysis of Four Groups Based on Estimated Value 74
4.30 Comparison of the Ratio of the Variation Between the Speed of the Five Techniques 79
4.31 The Comparative Analysis Between Group 1, Group 2, Group 3, Group 4 of Data Set 80
# LIST OF APPENDICES

<table>
<thead>
<tr>
<th>APPENDIX</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Create file</td>
<td>90</td>
</tr>
<tr>
<td>B</td>
<td>Implementation sorting techniques</td>
<td>95</td>
</tr>
</tbody>
</table>
CHAPTER 1

INTRODUCTION

1.1 Background of Study

Data structure is one of the most important techniques for organizing large data. It is considered as a method to systematically manage data in a computer that leads towards efficiently implementing different data types to make it suitable for various applications (Chhajed et al., 2013). In addition, data structure is considered as a key and essential factor when designing effective and efficient algorithms (Okun et al., 2011; Black, 2009). There are various studies that demonstrate the importance of data structures in software design (Yang et al., 2011; Andres et al., 2010).

So far, several researchers have focused on how to inscribe and improve the algorithm and ignoring data structures, while the data structures significantly affect the performance and the efficiency of the algorithm (Al-Kharabsheh et al., 2013). Sorting is one of the basic function of computer processes which has been widely used on database systems. A sorting algorithm is the arrangement of data elements in some order either in ascending or descending order. The algorithm can also be helpful to group the data on the basis of a certain requirement such as Postal code or in the classification of data into certain age groups. Thus, sorting is a rearrangement of items in a requested list dedicated in order to produce solution for a desired result. A number of sorting algorithms have been developed such as Quick sort, Merge sort, Insertion sort and Selection sort (Andres et al., 2010).

Meanwhile, several efforts have been taken to improve sorting techniques like Merge sort, Bubble sort, Insertion sort, Quick sort, Selection sort, each of them has a
different mechanism to reorder elements which increase the performance and efficiency of the practical applications and reduce the time complexity of each one. It is worth noting that when various sorting algorithms are being compared, there are a few parameters that must be taken into consideration, such as complexity, and execution time. The complexity is determined by the time taken for executing the algorithm (Goodrich, Tamassia & Mount, 2007). In general, the time complexity of an algorithm is generally written in the form of Big O(n) notation, where O represents the complexity of the algorithm and the value n represents the number of elementary operations performed by the algorithm (Jadoon et al., 2011).

Hence, this study investigates the efficiency of five sorting techniques, namely Selection sort, Insertion sort, Bubble sort, Quick sort, Merge sort and their behaviour on small and large data set. To accomplish these major tasks, proposed methodology comprises of three phases are introduced implementation of sorting technique, calculation of their complexity and comparative analysis. Each phase contains different steps and delivers useful results to be used in the next phase. After that, performance of these five sorting techniques were evaluated by three performance measures which are time complexity, execution time (run time) and size of dataset used.

1.2 Motivation

Despite the importance of data organization, sorting a list of input numbers or character is one of the fundamental issues in computer science. Sorting technique attracted a great deal of research, for efficiency, practicality, performance, complexity and type of data structures (Gurram & Jaideep, 2011). Therefore, data management needs to involved a certain sorting process (Liu and Yang, 2013). As a result, sorting is an important part in the data organization. Many researchers are attentive in writing the sorting algorithms but did not focus on the type of data structure used on them. Finding the most efficient sorting technique involves in examining and testing these techniques to finish the main task as soon as possible and identifying the most suitable structure for fast sorting and study the factors that affect the practical performance of each algorithm in terms of its overall run time. Thus, the aim of this study is to evaluate the efficiency of five sorting algorithms which are Bubble sort, Insertion sort,
Selection sort, Merge sort and Quick sort using static array data structure and to compare and analyse the results based on their runtime and complexity.

### 1.3 Research Objectives

Based on the research background, the three objectives of this research are as follows:

- i. implement five sorting techniques, namely Bubble sort, Insertion sort, Selection sort, Merge sort and Quick sort using static array data structure
- ii. calculate the complexity of the implemented sorting techniques as in (i)
- iii. compare and analyse result on the time taken and efficiency for the five sorting techniques based on the algorithms’ complexity.

### 1.4 Research Scope

This thesis focuses only on testing the effectiveness of five different sorting techniques, namely Bubble sort, Insertion sort, Selection sort, Merge sort and Quick on four groups of datasets with various data sizes which are 100 to 1,000 (Group 1), 2,000 to 10,000 (Group 2), 11,000 to 20,000 (Group 3) and 21,000 to 30,000 (Group 4). The performance of these five sorting techniques is evaluated by three performance measures which are time complexity, execution time (run time) and size of datasets.

### 1.5 Thesis Outline

This thesis outlines the background study of this research project, focusing on five different sorting algorithms. The motivation and the scope of the research are also presented. Chapter 2 discusses the literature review on static array data structure and sorting techniques. Chapter 3 presented the methodology of this research while Chapter 4 explains the implementation and detailed steps of the work. Finally, Chapter 5 concludes with an elaboration of the research achievements and future work.
CHAPTER 2

LITERATURE REVIEW

2.1 Overview

The overall goal of this chapter is to establish the significance of the general field of study. First, an overview of data structures with its main activities are presented and followed by the description of sorting techniques. It also covers some evaluation measures such as execution time, algorithm complexity and the use of Least Squares Regression for finding estimation values among various groups and ends with a summary of the chapter.

2.2 Data Structures

Data structure is a systematic organization of information and data to enable it to be used effectively and efficiently especially when managing large data. Data structures can also be used to determine the complexity of operations and considered a way to manage large amounts of data such as the index of internet and large corporate databases (Chhajed et al., 2013). According to Okun et al. (2011), an effective and efficient algorithm is required when designing highly efficient data structures. In the realm of software design, there are several studies that have attested the importance of data structures (Yang et al., 2011; Andres et al., 2010; Okun et al., 2011).

Data structures are generally based on the ability of a computer to fetch and store data at any place in its memory, specified by a pointer with a bit stringer
presenting a memory address. Thus, the data structures are based on computing the addresses of data items with arithmetic operations, (Okun, et al., 2011).

Furthermore, there are two types of data structures which are static data structure such as array and dynamic data structure such as linked list. Static data structure has a fixed size and the elements of static data structures have fixed locations. But in dynamic data structures, the elements are added dynamically. Therefore, the locations of elements are dynamic and determined at runtime (Yang et al., 2011).

2.3 Static Array Data Structures

An array is the arrangement of data in the form of rows and columns that is used to represent different elements through a single name but different indicators, thus, it can be accessed by any element through the index (Andres et al., 2010). Arrays are useful in supplying an orderly structure which allows users to store large amounts of data efficiently. For example, the content of an array may be changed during runtime whereas the internal structure and the number of the elements are fixed and stable (Yang et al., 2011).

An array could be called fixed array because they are not changed structurally after they are created. This means that the user cannot add or delete to its memory locations (making the array having less or more cells), but can modify the data it contains because it is not change structurally. Andres et al., (2010) illustrated that there are three ways in which the elements of an array can be indexed.

i. zero-based index. It is the first element of the array which is indexed by subscript 0.

ii. one-based indexing. It is the first element of the array which is indexed by the subscript 1.

iii. n-based indexing. It is the base index of an array which can be freely chosen.

Therefore, arrays are important structures in the computer science, because they can store a large amount of data in a proper manner while comparing with the list structure, which are hard to keep track and do not have indexing capabilities that makes it weaker in terms of structure (Yang et al., 2011).
2.3.1 Advantages of using Array

Each data structure has the weakness points and strength points. Listed below are advantages of array as mentioned by Andres et al. (2010):

i. arrays allow faster access to any item by using the index.
ii. arrays are simple to understand and use.
iii. arrays are very useful when working with sequences of the same type of data.
iv. arrays can be used to represent multiple data items of same type by using only single name but different indexes.

2.3.2 Disadvantages of using Array

Even though arrays are very useful data structures, however, there are some disadvantages as mentioned by Andres et al. (2010) which are listed below:

i. array items are stored in neighbouring memory locations, sometimes there may not be enough memory locations available in the neighbourhood.
ii. the elements of array are stored in consecutive memory locations. Therefore, operations like add, delete and swap can be very difficult and time consuming.

2.4 Execution Time

Execution time is the time taken to hold processes during the running of a program. The speed of the implementation of any program depends on the complexity of a technique or algorithm. If the complexity is low, then the implementation is faster, whereas when the complexity is high then the implementation is slow (Puschner & Koza, 1989). Keller (2000) argues that the execution time is the time for a program to process a given input. Keller believes that time is one of the important computer resources for two reasons: the time spent for the solution and the time spent for program implementation and for providing services.
2.5 Running Time Analysis

Running time is a theoretical process to calculate the approximate running time of a technique. For example, a program can take seconds or hours to complete an execution, depending on a particular technique used to lead the program (Bharadwaj & Mishra, 2013). Moreover, the runtime of a program describes the number of operations it executes during implementation and also the execution time of a technique. Furthermore, it should look forward to the worst case, an average case and best case performance of the technique. These definitions support the understanding of techniques complexity as mentioned by Bharadwaj & Mishra (2013).

2.6 Time Complexity

According to Estakhr (2013), the time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function with the length of a string representing the input. The time complexity of an algorithm is commonly expressed using Big(O) notation, which excludes coefficients and lower order terms. When expressed this way, the time complexity is said to be described asymptotically as the input size goes to infinity. The time complexity is commonly estimated by counting the number of elementary operations performed by the algorithm, where an elementary operation takes a fixed amount of time to perform. Thus, the amount of time taken and the number of elementary operations performed by the algorithm differ by at most a constant factor (Michael, 2006).

Time can mean the number of memory accesses performed, the number of comparisons between integers, the number of times some inner loop is executed, or some other natural unit related to the amount of real time the algorithm will take. The research tries to keep this idea of time separated from clock time, since many factors unrelated to the algorithm itself can affect the real time such as the language used, type of computing hardware, the proficiency of the programmer and optimization used by the compiler. If the choice of the units is wise, all of the other factors will not matter to get an independent measure of the efficiency of the algorithm. The time complexities of the algorithms studied are shown in Table 2.1.
Table 2.1: Time Complexity of Sorting Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time complexity</th>
<th>Best case</th>
<th>Average case</th>
<th>Worst case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bubble Sort</td>
<td>O(n)</td>
<td>O(n²)</td>
<td>O(n²)</td>
<td>O(n²)</td>
</tr>
<tr>
<td>Insertion Sort</td>
<td>O(n)</td>
<td>O(n²)</td>
<td>O(n²)</td>
<td>O(n²)</td>
</tr>
<tr>
<td>Selection Sort</td>
<td>O(n²)</td>
<td>O(n²)</td>
<td>O(n²)</td>
<td>O(n²)</td>
</tr>
<tr>
<td>Quick Sort</td>
<td>O(n²)</td>
<td>O(n²)</td>
<td>O(n.log(n))</td>
<td>O(n.log(n))</td>
</tr>
<tr>
<td>Merge sort</td>
<td>O(n.log(n))</td>
<td>O(n.log(n))</td>
<td>O(n.log(n))</td>
<td>O(n.log(n))</td>
</tr>
</tbody>
</table>

2.6.1 Worst-Case Analysis

The worst case analysis anticipates the greatest amount of running time that an algorithm needed to solve a problem for any input of size n. The worst case running time of an algorithm gives us an upper bound on the computational complexity and also guarantees that the performance of an algorithm will not get worse (Szirmay & Márton, 1998).

2.6.2 Best-Case Analysis

The best case analysis expects the least running time the algorithm needed to solve a problem for any input of size n. The running time of an algorithm gives a lower bound on the computational complexity. Most of the analysts do not consider the best case performance of an algorithm because it is not useful (Szirmay & Márton, 1998).

2.6.3 Average Case Analysis

Average case analysis is the average amount of running time that an algorithm needed to solve a problem for any input of size n. Generally, the average case running time is considered approximately as bad as the worst case time. However, it is useful to check the performance of an algorithm if its behaviour is averaged over
all potential sets of input data. The average case analysis is much more difficult to carry out, requiring tedious process and typically requires considerable mathematical refinement that causes worst case analysis to become more prevalent (Papadimitriou, 2003).

2.7 Big-O Notation

Big-O notation is used to characterize upper bound of a function that states the maximum value of resources needed by an algorithm to do the execution (Knuth, 1976). According to Black (2007), Big(O) notation has two major fields of application namely mathematics and computer science. In mathematics, it is usually used to show how closely a finite series approximates a given function. In computer science, it is useful in the analysis of algorithms. There are two usages of Big (O) notation which are infinite asymptotic and infinitesimal asymptotic. This singularity is only in the application and not in precept; however, the formal definition for the Big(O) is the same for both cases, only with different limits for the function evidence.

Let f(n) and g(n) be functions that map positive integers to positive real numbers. Say that f(n) is O(g(n)) (or if \( n \in O(g(n)) \)) if there exists a real constant \( c > 0 \) and there be an integer constant \( n_0 \geq 1 \) such that \( f(n) \leq c \cdot g(n) \) for every integer \( n \geq n_0 \) as shown in Figure 2.1.

Figure 2.1: Big O Notation Graph.
2.8 Sorting

Sorting is the process of rearranging the set of elements which can be sorted alphabetically, descending or increasing order; or based on a certain attribute such as city population, area, or zip code (Coxon, 1999).

According to Horan & Wheeless (1977), there are many sorting algorithms that have been developed and analysed throughout the literature. This indicates that sorting is an important area of study in computer science. Sorting a large number of items can take a substantial amount of computing resources. The efficiency of a sorting algorithm is based on the number of items being processed. For small collections, a complex sorting method may not be as useful compared to a simpler sorting method. This section discusses sorting techniques and compares them with respect to their running time. Sorting algorithm is one of the most basic research areas in computer science. The aim is to make data easier to be updated. Sort is a significant process in computer programming. It can be used to sort sequence of data by an ordering procedure using a type of keyword. The sorted sequence is most helpful for later updating activities such as search, insert and delete.

2.8.1 Quick Sort

Pooja (2013) stated that quick sort is the fastest internal sorting algorithm among other developed algorithms. Unlike merge sort, quick sort needs less memory space for sorting an array. Therefore, it is vastly used in most real time applications with large data sets. Quick sort uses divide and conquer method for solving problems. It works by partitioning an array into two parts, then sorting the parts independently. It finds the elements called pivot which divides the array into halves in such a way that elements in the left half are smaller than the pivot, and elements in the right half are greater than pivot.

The algorithm repeats this operation frequently for both the sub arrays. In general, the leftmost or the rightmost element is selected as a pivot. Selecting the leftmost and rightmost element as pivot was practiced since in the early version of quick sort (Pooja, 2013). Quick sort has a fast sorting algorithm on the time complexity of O(n log n) in contrast to other developed algorithms. However,
selecting the leftmost or rightmost element as a pivot causes the worst-case running
time of O (nlogn) when the array is already sorted.

2.8.2 Selection Sort

Selection sort is notable for its programming simplicity and in certain situation can
over perform other sorting algorithms. It works by finding the smallest or highest
element from the unsorted list and swaps with the first element in the sorted list and
then finds the next smallest element from the unsorted list then swap with the second
element in the sorted list. The algorithm continues this operation until the list being
sorted (Bharadwaj & Mishra, 2013). Selection sort requires a constant amount of
memory space with only the data swaps within the allocated spaces. However, like
some other simple sorting methods, Selection sort is also inefficient for large
datasets or arrays (Bharadwaj & Mishra, 2013). Al-Kharabsheh, et at. (2013) stated
that Selection sort has O(n^2) time complexity, making it inefficient on large lists and
performs worse than the similar Insertion sort but better than Bubble sort.

2.8.3 Insertion Sort

Insertion sort is a simple and efficient sorting algorithm, beneficial for small size of
data. It works by inserting each element into its suitable position in the final sorted
list. For each insertion, it takes one element and finds the suitable position in the
sorted list by comparing with contiguous elements and inserts it in that position
(Ching, 1996). This process is iterative until the list is sorted in the desired order.
Unlike other sorting algorithms, Insertion sort go through the array or list only once,
requiring only a constant amount of memory space as the data is sorted within the
array itself by dividing itself into two sub-array, one for sorted and one for unsorted.
However, it becomes more inefficient for a greater size of input data when compared
to other algorithms.
2.8.4 Bubble Sort

Bubble sort is a simple and the slowest sorting algorithm which works by comparing each element in the list with progress elements and swapping them if they are in undesirable order. The algorithm continues this operation until it makes a pass right through the list without swapping any elements, which shows that the list is sorted. This process takes a lot of time and especially slow when the algorithm works with a large data size. Therefore, it is considered to be the most inefficient sorting algorithm with large dataset (Astrachan, 2003).

According to Batcher (1968), Bubble sort has a complexity of $O(n^2)$, where $n$ indicates the number of elements to be sorted. However, although other simple sorting algorithms such as Insertion sort and Selection sort have the same worst case complexity of $O(n^2)$, the efficiency of Bubble sort is relatively lesser than other algorithms.

2.8.5 Merge Sort

According to Mehlhorn (2013), Merge sort has a complexity of $O(n \log n)$. The $O(n \log n)$ worst case upper bound on merge sort stems from the fact that merge is $O(n)$. The application of the Merge sort produces a stable sort, which means that the applied preserves the input order of equal elements in the sorted output. Merge sort, invented by Von Neumann and Morgenstern (1945) works by divide and conquer method and it is based on the division of the array into two halves at each stage and then goes to a compare stage which finally merges these parts into one single array. This is also a comparison-based sorting algorithms such as Bubble sort, Selection sort and Insertion sort. In this method, the array is divided into two halves. Then recursively sort these two parts and merge them into a single array. When working with small array, Merge sort is not a good choice as it requires an additional temporary array to store the merged elements with $O(n)$ space.

2.9 Simple Linear Regression

Regression analysis is a statistical function used to find the estimated value between variable groups, which includes many of the techniques that are used in
special preparations analysed to determine the relationship between the dependent variable and independent variable (Armstrong, 2012).

It is a least square estimator of a linear regression model with a single explanatory variable. In other words, it is a simple straight line which passes through a series of dots that make total residuum any distance between the real point and the estimated point.

In general, the regression model provides estimated value of conditional expectation of the dependent variable given the independent variables. The goal of estimation value is called the regression function. In regression analysis, it is also motivating to describe the variation of the dependent variable around the regression function which can be described by a probability distribution. Regression analysis used to predict or find a relationship between the independent variable and dependent variable moreover, its impact on the dependent variable. Thus regression analysis finds a causal relationship between the variables. The linear regression equations are as mentioned by Waegeman et al. (2008):

\[ Y = a + b \times X \]  
(1)

Where
\[ Y : \] is the dependent variable
\[ X : \] is the independent variable
\[ a : \] is the constant (or intercept)
\[ b: \] is the slope of the regression line

The equation of squares regression is

\[ SR = \left\{ \frac{1}{N} \times \sum \left[ (x_i - \bar{x}) \times (y_i - \bar{y}) \right] / (\sigma_x \times \sigma_y) \right\}^2 \]  
(2)

\[ \sigma_x = \sqrt{\sum (x_i - \bar{x})^2 / N} \]
\[ \sigma_y = \sqrt{\sum (y_i - \bar{y})^2 / N} \]

Where
\[ N: \] is the number of observations used to fit the model.
\[ \sum: \] is the summation symbol
\[ Xi: \] is the x value of observation i
\[ Yi: \] is the y value of observation i
\[ Y: \] is the mean y value
\[ \sigma_x: \] is the standard deviation of x
\( \sigma_y \): is the standard deviation of \( y \)

In mathematics, a ratio is a relationship between two numbers indicating how many times the first number contains the second. The equation of ratio according to Ching (1996):

\[
Speed \approx \frac{\text{estimation value technique 1}}{\text{estimation value technique 2}}
\]  

(3)

### 2.10 Related Work

A considerable amount of literature has been published on sorting techniques. While looking into large and growing body of literature, it is appeared that sorting techniques have been proven to be successful for data structures. Thus, the data structures have an impact on the efficiency of these sorting techniques (Ching, 1969). Al-Kharabsheh et al. (2013) discussed and reviewed the performance of sorting techniques where comparison of the algorithms were based on the time of implementation. It was found that for small data, the six techniques perform well, but for large input data, only Quick sort and GCS sort are considered fast. Pooja (2013) examined several sorting algorithms and discussed the performance analysis of these sorting algorithms based on their complexity while testing them with list data structure. It was found that the merge sort and quick sort have high complexity but faster in large lists.

In the work of Chhajed, Uddin & Bhatia (2013), four techniques which are Insertion sort, Quick sort, Heap sort and Bubble sort were compared. Although all these techniques are of \( O(n^2) \) complexity, it was found that they produced different results in execution time with Quick sorting technique being the most efficient in terms of execution time. Bharadwaj & Mishra (2013) discussed the four sorting algorithms - Insertion sort, Bubble sort, Selection sort and Merge sort. They designed a new sorting algorithm named Index sort to check the performance of these sorting algorithms, then compared it with other four sorting techniques based on their run time and found that the Index sort is faster than the other sorting algorithms. Table 2.2 provides a summaries of some selected studies on sorting algorithms.
Table 2.2: Summary of some Selected Studies on Sorting Algorithms.

<table>
<thead>
<tr>
<th>Author</th>
<th>Title</th>
<th>Technique</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pooja (2013)</td>
<td>Comparative Analysis &amp; Performance of Different Sorting Algorithm in Data Structure</td>
<td>Bubble Sort, Selection Sort, Insertion Sort, Merge Sort and Quick Sort using list data structure</td>
<td>Merge Sort and Quick Sort are more complicated, but faster than other techniques</td>
</tr>
<tr>
<td>Al-Kharabsheh et al. (2013)</td>
<td>Review on Sorting Algorithms A Comparative Study</td>
<td>Comparing the Grouping Comparison Sort (GCS), Selection sort, Quick sort, Insertion sort, Merge sort and Bubble based on execution time</td>
<td>For small data, the six techniques are performing well, but for large input, Quick sort and GCS sort are fast.</td>
</tr>
<tr>
<td>Chhajed, Uddin &amp; Bhatia (2013)</td>
<td>A Comparison Based Analysis of Four Different Types of Sorting Algorithms in Data Structures with Their Performances</td>
<td>insertion sort, quick sort, heap sort, and bubble sort, time complexity to reach our conclusion</td>
<td>the four sorting techniques Insertion, Heap, bubble and Quick sort techniques give the result of the order of $N^3$ but Quick sorting technique will be more helpful than other techniques</td>
</tr>
<tr>
<td>Ching (1996)</td>
<td>A comparison E study of linked List sorting technique</td>
<td>The sediment sort, quick sort, merge sort, tree sort, selection sort, bubble sort using dynamic data structure linked lists,</td>
<td>the sediment sort is the slowest algorithm for sorting linked lists and the O(nlogn) group performs much better than the O($n^2$) group</td>
</tr>
<tr>
<td>Bharadwaj &amp; Mishra (2013)</td>
<td>Comparison of Sorting Algorithms based on Input Sequences</td>
<td>Insertion Sort, Bubble Sort, Selection Sort, Merge Sort and index sort</td>
<td>The index sort faster than other sorting techniques</td>
</tr>
</tbody>
</table>
2.11 Summary

A literature review on sorting algorithms is presented to provide a background information related to the scope of the dissertation, encompassing two technical areas namely data structure and sorting techniques. The theoretical aspects of data structure, static array data structures and their advantages and disadvantages are presented. The complexity of these sorting algorithms, based on Big-O notation are also discussed, examining their performance with small and large datasets. The large body of related work presented in this chapter helps in understanding the sorting algorithms implemented in this research. Chapter 3 explains the process map and main steps involved in the research methodology of this study.
CHAPTER 3

RESEARCH METHODOLOGY

3.1 Overview

This chapter discusses the methodology of this project. Firstly, it presents the preliminary stage of this research by introducing the proposed framework. This research focuses on five sorting techniques and applying them to static array data structure which consists of four groups of data sets. Then, the research activities and all main phases of this project are discussed. Finally, it presents a summary of this chapter.

3.2 Proposed Framework

The proposed framework involved three phases, namely implementation of sorting technique, calculating the complexity and comparative analysis as shown in Figure 3.1. The first phase is implementation of five sorting techniques, which are Bubble sort, Insertion sort, Selection sort with \( O(n^2) \) complexity, Merge sort and Quick sort with \( O(n \log n) \) complexity. The second phase is calculating the complexity of these five sorting techniques. The third phase is comparing and analysing theses sorting techniques with performance measurement execution time per second, and size of the data set, based on simple linear regression.
Figure 3.1: The Three Phases of the Study.

Figure 3.2 depicts the process map which comprises of all three phases, where each phase contains its different steps and delivers results to be used in the next phase.
Figure 3.2: Process Map.

The explanations of the phases are discussed in the following section.
3.3 Phase 1: Implementing the Sorting Techniques

Phase 1 encompasses the implementation of five sorting techniques. The five sorting techniques are Bubble sort, Insertion sort, Selection sort with $O(n^2)$ complexity, Merge sort and Quick sort with $O(n \log n)$ complexity using C++ programming language respectively. The input data are random integers between 100 and 30,000. It is further divided into four groups in order to ease the analysis process. Group 1 consists of 100 to 1,000, group 2 consists of 2,000 to 10,000, group 3 consists of 11,000 to 20,000 and group 4 consists of 21,000 to 30,000. Data processing is done on the same computer.

The implementation of each case study datasets are grouped into six phases. Firstly, datasets are grouped by creating a new file. Secondly, random integer numbers are read, subsequently in the third phase, these integer numbers are written in the created file. In fourth phase, the integer numbers are inserted in an array and in the fifth phase, choice menu() function is used to represent the list of choices for different sorting techniques which is followed by the last phase where the datasets are used in the clock function to calculate the run time.

3.4 Phase 2: Calculating the Complexity of Sorting Techniques

In this phase, the sorting algorithms are tested using four groups with different data sizes. Then, the program calculate the complexity of each sorting technique which are Bubble sort, Insertion sort, Selection sort, Merge sort and Quick sort using Big O (O) concept. This concept is used to measure the complexity of algorithms and it is useful in the analysis of algorithms for efficiency.

3.5 Phase 3: Comparative Analysis

In this phase, the sorting algorithms are tested again using the four data groups of different sizes. However, this time, they are tested in terms of their efficiency based on the complexity, execution time per second and size of input data. The performance measure is analyzed on two different behaviours which are $O(n^2)$ and $O(n \log n)$. In order to perform the required analysis, linear regression is used and estimated value is calculated by using Excel. The reason for using linear regression
is that, it can help in finding fitting linear regression and square regression to grasp how the ideal value of the dependent variable (execution time) changes when any one of the independent variables (number of elements) is varied, while the other independent variables are still stable based on Equation 2, stated in Section 2.9.

Thus, Regression analysis is used to find the estimated value of the variable given the independent variable. The details of each component are explained further in Chapter 4. The results are compared and analyzed after they were obtained from the implementation of the sorting techniques and the analysis, based on three measurements with complexity, execution time per second and input data set of four groups of data by linear regression. In this quantitative comparison, estimated value is used because it can find fitting linear regression for better overall performance. The estimation value and ratio value are considered as comparison criteria between sorting technique that have $O(n \log n)$ and $O(n^2)$ to determine which one is more efficient. In doing the comparison among these sorting techniques, this work will examine:

i. the estimated value for each sorting technique and for each group of dataset based on equation 1 as stated in Chapter 2, Section 2.9.

ii. the average for estimation value of five sorting techniques.

iii. the average of the ratio between the sorting techniques based on equation 3 as illustrated in Chapter 2, Section 2.9.

iv. the average of speed ratio of sorting technique between four groups based on equation 3 shown in Chapter 2, Section 2.9.

### 3.6 Summary

Chapter 3 illustrates the methodology of the research starting from creating the dataset used in the experiment until the evaluation of the experimental results. The methodology presented includes three phases namely implementation of sorting technique, calculation of algorithm complexity and comparative analysis of these sorting technique on four different sizes of the dataset. It is also discussed how the result are analyzed and compared.
CHAPTER 4

IMPLEMENTATION AND ANALYSIS

4.1 Overview

This chapter presents the results of the implemented sorting algorithms which are Bubble sort, Insertion sort, Selection sort, Merge sort and Quick sort, and a comparative analysis is carried out based on the results. The outcome of this study is discussed in detail based on the methodology presented in Chapter 3. This chapter is divided into two sections. The first section gives the details about the implementation of these sorting techniques by using four groups of data sets, in the form of arrays. Later, the complexity is calculated based on Big-O concept and measurements in time execution per second were taken based on the function of clock time in the C++ program. The second section illustrates the comparative analysis of these sorting techniques from each data set group of each sorting algorithm. All the sorting algorithms are divided into two groups – first group using complexity level $O(n^2)$ which includes Bubble, Insertion and Selection and second group using complexity level $O(n \log n)$ which includes Merge and Quick based on simple linear regression analysis technique.

4.2 Implementation of Sorting Techniques

All the five sorting algorithms, namely Bubble sort, Selection Sort, Insertion sort, Merge sort and Quick sort are implemented in C++ programming language based on arrays of case study data set from 100 to 30,000. After that, all five sorting algorithms were tested for random sequence input data set of length between 100 and
30,000. All the five sorting algorithms were executed on machine with Operating System equipped with Intel (R) Core (TM) 2 Duo CPU E8400 @ 3.00 GHz (2 CPUs) and installed memory (RAM) of 2038 MB. The CPU time was taken in per second. The executed data set to the static data structure were as illustrated in the following subsections.

4.2.1 Description of Test Data Sets

A data set is a random integer number with several files of integers, selected at random to be used to test the five sorting methods. The files are of different sizes, ranging between 100 and 30,000. Then the number used for these data set are divided to four groups of different intervals as shown in Table 4.1.

<table>
<thead>
<tr>
<th>Group</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group 1</td>
<td>100-1,000</td>
</tr>
<tr>
<td>Group 2</td>
<td>2,000-10,000</td>
</tr>
<tr>
<td>Group 3</td>
<td>11,000-20,000</td>
</tr>
<tr>
<td>group4</td>
<td>21,000-30,000</td>
</tr>
</tbody>
</table>

4.2.2 Create and Open File

In this section, the data set is created randomly by generating the number of integers and stored in a variable called number. Then the number is saved in an output file. This process is continued until the end of statement for loop of integer numbers as shown in Table 4.2. However, this process runs only once, without repetition with complexity level of O(n).
Table 4.2: Create and Open File Complexity

<table>
<thead>
<tr>
<th>Step</th>
<th>Big O</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create and open the file</td>
<td>O(1)</td>
</tr>
<tr>
<td>ofstream outfile1(&quot;numbers1.txt&quot;,ios::out);</td>
<td></td>
</tr>
</tbody>
</table>
| for(i=0;i<100;i++){
  number=rand()%1000;
  outfile1<<number<<endl;
}(outfile1.close(); | O(n)    |
| Total                                     | O(n)    |

4.2.3 Insert Data File in to Array

This section presents the insert data file as array. The complexity level is \(O(n)\) and when it is closed, the process is repeated only once. Thus the total complexity is \(O(n+1) = O(n)\). Table 4.3 shows the complexity of inserting data file into array.


Ching Kuang Shene1Michigan Technological University, Department of Computer Science Houghton A Comparative Study of LinkedList Sorting Algorithms by, MI 49931-1295shene@mtu.edu 1996.


keller, Complexity. [http://www.cs.hmc.edu](http://www.cs.hmc.edu)


Miss. Pooja K. Chhatwani, Miss. Jayashree S. Somani, 2013. Comparative Analysis & Performance of Different Sorting Algorithm in Data Structure Lecturer. Information Technology, Dr. N. P. Hirani Institute of Polytechnic. Pusad,Maharashtra, India


