**Student No**: 1118676

**Student** **Name**: Naomi Leech

**Module** **Code**: CM2303  
  
**Module Title**: Algorithms and Data Structures

**Coursework Title**: Run time analysis of bubble sort and quick sort

**Lecturer**: Dr Christine Mumford

**Hours Spent on this Exercise**:

**Special Provisions**: -

Run Time Analysis of Bubble Sort and Quick Sort

Contents

Introduction pg

Design

Bubble Sort Pseudocode pg

Quick Sort Pseudocode pg

Implementation pg

Testing pg

Experimental Method

Problems Encountered pg

Results

Bubble Sort pg

Quick Sort pg

Conclusion pg

Introduction

In this report, I will undertake an empirical study of two algorithms (bubble sort and quick sort), in which I will run programs to compare the run time and then measure, record and analyse the timings of each. This will be done to find out which is the better sorting algorithm. Both algorithms have different run-time complexities which can be defined in three categories: Best, Worst and Average Case. The complexities I expect for each will be as follows:

Bubble Sort

*Best Case:* O(n)

*Worst Case:* O(n2)

*Average Case:* O(n2)

Quick Sort

*Best Case:* O(n log n)

*Worst Case:* O(n2)

*Average Case:* O(n log n)

The codes will be written in Python, and the results written then opened up as a csv file in Gnumeric. This will allow me to create graphs to compare easily the complexities of each algorithm, and ensure they work and the graphs look as they are predicted by theory. I will run the codes for different sized arrays of integers.

Design (Pseudocode)

Before starting the implementation I will write out the two algorithms in pseudocode. I will use pseudocode as it will easily explain what each line in the code is doing, as well as help me to implement it later on in the implementation stage of this assignment.

*Bubble Sort*

Algorithm BubbleSort(n, A)

Input: An array, A, of numbers of length n

Output: The array, A sorted

*Quick Sort*

Algorithm QuickSort()

Input: an array, A, of numbers of length n

Output: The array, A sorted

Implementation

I acquired the following code for the algorithms off a fellow course-mate, who in turn acquired it from the internet. I changed the variable names throughout the code, as I used it as a method to ensure I understood and knew what each bit in the code did.

*Bubble Sort*

*Quick Sort*

Testing

Experimental Method

The best case for the algorithms is when the list is already sorted. The reason it is the best case as it will take the quickest time to sort the list, as the integers are already in the correct ascending order.

The average case for the algorithms is the average time it will take to sort an unsorted list.

The worst case for the algorithms is when the list is reversely sorted. The reason it is the worst case is because it will take the longest time to sort the list into the correct ascending order.

*Problems encountered:*

One of the problems encountered when implementing and running the code was trying to get the results to be input into 2 columns in the spreadsheet. This was overcome by opening it in a different programme. Initially Excel was being used, and the problem was solved by using Gnumeric instead.

Results

Plotting graphs is the best way to analyse the run time for the algorithms. I have plotted the complexity (e.g. n2) along the x-axis and the time along the y-axis.

The graphs are as follows:

*Bubble Sort*

*Best Case:*

|  |  |
| --- | --- |
|  |  |
|  |  |

*[GRAPH]*

*Average Case:*

|  |  |
| --- | --- |
|  |  |
|  |  |

[GRAPH]

*Worst Case:*

|  |  |
| --- | --- |
|  |  |
|  |  |

*[GRAPH]*

*Quick Sort*

*Best Case:*

|  |  |
| --- | --- |
|  |  |
|  |  |

*[GRAPH]*

*Average Case:*

|  |  |
| --- | --- |
|  |  |
|  |  |

*[GRAPH]*

*Worst Case*:

|  |  |
| --- | --- |
|  |  |
|  |  |

*[GRAPH]*

Conclusion