A red sign with white text

Description automatically generated

BSc Computer Science

**Module Title**

Advanced Algorithms

**Assessment Title**

Mini Report

**Assessment Weighting**

5% of the module mark

**Student Name**

Reece Turner

(09/05/2024)

Contents

[1. Tasks 3](#_Toc166250654)

[1.1 Password Generator 3](#_Toc166250655)

[1.2 Longest Substring 3](#_Toc166250656)

[1.3 Parallelism Programming 3](#_Toc166250657)

[1.4 Cheapest Train Tickets 3](#_Toc166250658)

[2. Appendix 4](#_Toc166250659)

[Figure 1: Password Generator 4](#_Toc166250660)

[Figure 2: Longest Substring 5](#_Toc166250661)

[Figure 3: Parallelism Programming 6](#_Toc166250662)

[Figure 4: Cheapest Train Tickets 6](#_Toc166250663)

# 1. Tasks

## 1.1 Password Generator

For this task I had to write an algorithm that generates passwords within a rule set. In **Figure 1** my rule set can be seen within the flowchart’s algorithm decision statements. To perform such decisions for each password I implemented Tail Recursive Optimisation (TCO). This algorithm effectively calls its entry point until a task is complete. In contrast to typical recursion, TCO will handle memory leaks by ending the recursion if criterion is met to prevent overflow. However, given the size of the input, , and the number of categories, , the time complexity of this algorithm would be therefore expressing exponentiality showing that smaller inputs are an optimal use case.

A computer screen with white text

Description automatically generatedA screenshot of a graph

Description automatically generated

*Figure 1.1: Execution of the program with password length of 4.*

In **Figure 1.1** the algorithm demonstrated complexity by averaging the time over 5 runs on a password length of 4. You can see through the numbers its permuting through each result in sequence to find the product. As a result, it can be slower, especially ran in a Python environment; however, this method was more effective over implementing the “itertools” library ‘product’ due to its built-in functionality not being able to consider rule sets.

The libraries for this task where built-in so the user should be able to run out of the box. These consist of OS to find the task file, TIME to record algorithm duration and TYPING to define function return data types.

## 1.2 Longest Substring

In this task I chose the implement dynamic programming with a sliding window approach. I went with dynamic programming over other approaches such as quick sort because the algorithm uses subroutines to solve the larger problem at hand by storing and reusing previous elements – this can be seen in the **Figure 2** flow chart. In our case the sliding window iterated through the string file and search for possible longest substrings and where the invalid point is discovered become the algorithm starting point, effectively reducing complexity to using this approach.

A screenshot of a computer program

Description automatically generated

*Figure 2.1: Running the algorithm for different types of substring lengths.*

**Figure 2.1** demonstrates each repeated elements input being passed in and returning all possible substrings for that entry. Alternatively, if there are no substrings for the input given then a warning displays no valid substrings found, providing the user with some clarity into whats occurring at that time. Similarly to task 1, the algorithm requires built-in libraries accept with the additional use of CSV to read in file data rather than producing it via predefined arrays enabling the user to effectively use any means of data.

## 1.3 Parallelism Programming

## 1.4 Cheapest Train Tickets

# 2. Appendix

## Figure 1: Password Generator

A diagram of a company

Description automatically generated

## Figure 2: Longest Substring

A diagram of a software system

Description automatically generated

## Figure 3: Parallelism Programming

## Figure 4: Cheapest Train Tickets